Census Income Project

Importing necessary Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import os
import scipy as stats
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Importing the dataset

In [2]:
```
df= pd.read_csv('census_income.csv')
```

In [3]:
```
df
```

Out[3]:

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relations
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husb
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-fa
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husb
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	V
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	V
...	...	...	...	...	...	...	...	
32555	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	V
32556	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husb
32557	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmar
32558	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-c
32559	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	V

32560 rows × 15 columns

- So here we can observe that the dataset contains the details of the annual income of the persons .
- In the dataset "Income" is the target variable which seems to be having 2 classes so it will be termed to be a "Classification Problem" where we need to predict whether the income of the person is over $50k per year or not .
- The dataset contains both numerical and categorical columns.

In [4]:
```
df.head(20)
```

Out[4]:

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife
5	49	Private	160187	9th	5	Married-spouse-absent	Other-service	Not-in-family
6	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband
7	31	Private	45781	Masters	14	Never-married	Prof-specialty	Not-in-family
8	42	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband
9	37	Private	280464	Some-college	10	Married-civ-spouse	Exec-managerial	Husband
10	30	State-gov	141297	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband
11	23	Private	122272	Bachelors	13	Never-married	Adm-clerical	Own-child
12	32	Private	205019	Assoc-acdm	12	Never-married	Sales	Not-in-family
13	40	Private	121772	Assoc-voc	11	Married-civ-spouse	Craft-repair	Husband
14	34	Private	245487	7th-8th	4	Married-civ-spouse	Transport-moving	Husband
15	25	Self-emp-not-inc	176756	HS-grad	9	Never-married	Farming-fishing	Own-child
16	32	Private	186824	HS-grad	9	Never-married	Machine-op-inspct	Unmarried
17	38	Private	28887	11th	7	Married-civ-spouse	Sales	Husband
18	43	Self-emp-not-inc	292175	Masters	14	Divorced	Exec-managerial	Unmarried
19	40	Private	193524	Doctorate	16	Married-civ-spouse	Prof-specialty	Husband

Showing the first 5 rows of the dataset

Categorical Columns:

- Workclass
- Education
- Marital_status
- Occupation
- Relationship
- Race
- Sex
- Native_country
- Income

Numerical Columns: (continuous)

- Age
- Fnlwgt(Final Weight): sampling weight
- Education_num: Total number of years of education
- Capital_gain: Income from investment soures other than salary/wages
- Capital_loss: Income from investment sources other than salary/wages
- Hours_per_week

In the dataset we can observe some corrupted data which is filled as '?', so we can either drop this or we can fill this with some numbers.

Exploratory Data Analysis(EDA)

In [5]:
```
df.shape
```

Out[5]:
```
(32560, 15)
```

- The dataset contains 32560 rows and 15 columns .

Out of 15 columns 14 are independent features and remaining 1 is our target column that is 'Income'

In [6]:
```
df.dtypes
```

Out[6]:
```
Age                int64
Workclass         object
Fnlwgt             int64
Education         object
Education_num      int64
Marital_status    object
Occupation        object
Relationship      object
Race              object
Sex               object
Capital_gain       int64
Capital_loss       int64
Hours_per_week     int64
Native_country    object
Income            object
dtype: object
```

- The dataset contains 2 types of data namely integer type and object type.

We will convert this object type data into numerical using encoding techniquies before building the model.

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32560 entries, 0 to 32559
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Age             32560 non-null  int64 
 1   Workclass       32560 non-null  object
 2   Fnlwgt          32560 non-null  int64 
 3   Education       32560 non-null  object
 4   Education_num   32560 non-null  int64 
 5   Marital_status  32560 non-null  object
 6   Occupation      32560 non-null  object
 7   Relationship    32560 non-null  object
 8   Race            32560 non-null  object
 9   Sex             32560 non-null  object
 10  Capital_gain    32560 non-null  int64 
 11  Capital_loss    32560 non-null  int64 
 12  Hours_per_week  32560 non-null  int64 
 13  Native_country  32560 non-null  object
 14  Income          32560 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

- So here above cell gives the information about the dataset which includes indexing type, column type, no-null values and memory usage.

In [8]:
```
df.nunique().to_frame("No. of unique values")
```

Out[8]:

	No. of unique values
Age	73
Workclass	9
Fnlwgt	21647
Education	16
Education_num	16
Marital_status	7
Occupation	15
Relationship	6
Race	5
Sex	2
Capital_gain	119
Capital_loss	92
Hours_per_week	94
Native_country	42
Income	2

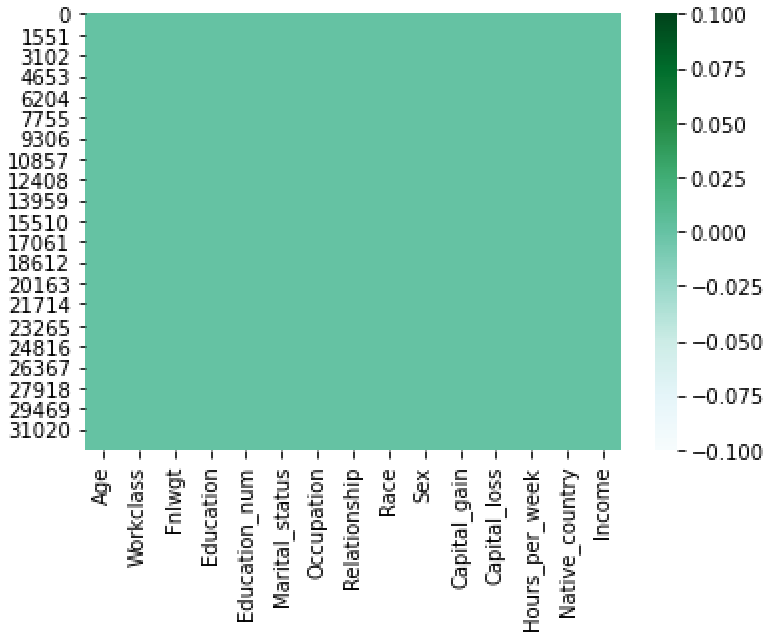
- So here we can see the number of unique values present in each columns

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Age               0
Workclass         0
Fnlwgt            0
Education         0
Education_num     0
Marital_status    0
Occupation        0
Relationship      0
Race              0
Sex               0
Capital_gain      0
Capital_loss      0
Hours_per_week    0
Native_country    0
Income            0
dtype: int64
```

- There is no missing values present in the dataset and data is cleaned.

In [10]:
```
# let's visualize the null values clearly
sns.heatmap(df.isnull(),cmap='BuGn')
plt.show()
```



- By visualizing we can say there are no missing values. So our data is cleaned.

In [11]:
```
df.columns
```

Out[11]:
```
Index(['Age', 'Workclass', 'Fnlwgt', 'Education', 'Education_num',
       'Marital_status', 'Occupation', 'Relationship', 'Race', 'Sex',
       'Capital_gain', 'Capital_loss', 'Hours_per_week', 'Native_country',
       'Income'],
      dtype='object')
```

These are the columns present in the dataset

Let's check the counts of each column to know which columns has ? sign and will take care of it.

Value Count Function

Let's check the list of value counts in each columns to find if there are any unexpected or corrupted entries in the dataset.

In [12]:
```
for i in df.columns:
    print(df[i].value_counts())
    print('==========================================================')
```

```
36    898
31    888
34    886
23    877
35    876
     ...
83      6
85      3
88      3
87      1
86      1
Name: Age, Length: 73, dtype: int64
==========================================================
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 ?                    1836
 State-gov            1297
 Self-emp-inc         1116
```

- Here the columns capital gain and capital loss have more than 90% of zeroes and it is not required for our prediction. Let's drop them

In [13]:
```
# droping the columns having more number of 0's
df.drop("Capital_gain",axis=1,inplace=True)
df.drop("Capital_loss",axis=1,inplace=True)
```

- The columns Workclass, Occupation and Native_country have '?' sign , it is not NAN value but we need to fill it.

Filling '?' Values

In [14]:
```
df['Workclass'] =df.Workclass.str.replace('?','Private')
df['Occupation'] =df.Occupation.str.replace('?','Prof-speciality')
df['Native_country'] =df.Native_country.str.replace('?','United-States')
```

- Now we have replaced the '?' values with mode .

In [15]:
```
df['Workclass'].value_counts()
```

Out[15]:
```
 Private             24532
 Self-emp-not-inc     2541
 Local-gov            2093
 State-gov            1297
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: Workclass, dtype: int64
```

In [16]:
```
df['Occupation'].value_counts()
```

Out[16]:
```
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3769
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 Prof-speciality      1843
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: Occupation, dtype: int64
```

In [17]:
```
df['Native_country'].value_counts()
```

Out[17]:
```
United-States                 29752
Mexico                          643
Philippines                     198
Germany                         137
Canada                          121
Puerto-Rico                     114
El-Salvador                     106
India                           100
Cuba                             95
England                          90
Jamaica                          81
South                            80
China                            75
Italy                            73
Dominican-Republic               70
Vietnam                          67
Guatemala                        64
Japan                            62
Poland                           60
Columbia                         59
Taiwan                           51
Haiti                            44
Iran                             43
Portugal                         37
Nicaragua                        34
Peru                             31
France                           29
Greece                           29
Ecuador                          28
Ireland                          24
Hong                             20
Cambodia                         19
Trinadad&Tobago                  19
Thailand                         18
Laos                             18
Yugoslavia                       16
Outlying-US(Guam-USVI-etc)       14
Honduras                         13
Hungary                          13
Scotland                         12
Holand-Netherlands                1
Name: Native_country, dtype: int64
```

- We can notice there are no '? ' sign in these columns means we have filled them.

- Most of the columns have unique type of classes, let's replace them with the new classes.

In [18]:
```
df.Workclass=df.Workclass.replace([' Local-gov',' State-gov',' Federal-gov' ],'
df.Workclass=df.Workclass.replace([' Private',' Self-emp-not-inc',' Self-emp-ir
df.Workclass=df.Workclass.replace([' Without-pay',' Never-worked'],'Not-working
```

In [19]:
```
df["Workclass"].value_counts()
```

Out[19]:
```
 Pvt-Sector     28189
 Govt-Sector     4350
Not-working        21
Name: Workclass, dtype: int64
```

In [20]:
```
df["Education"]=df["Education"].replace([' Preschool',' 1st-4th',' 5th-6th',' 7
df["Education"]=df["Education"].replace([' HS-grad',' Prof-school'],' High-Scho
df["Education"]=df["Education"].replace([' Some-college',' Assoc-voc',' Assoc-a
df["Education"]=df["Education"].replace([' Bachelors'],' Under_Graduation')
df["Education"]=df["Education"].replace([' Masters'],' Post_Graduation')
df["Education"]=df["Education"].replace([' Doctorate'],' PhD')
```

In [21]:
```
df["Education"].value_counts()
```

Out[21]:
```
 High-School         11077
 Secondary-School     9740
 Under_Graduation     5354
 School               4253
 Post_Graduation      1723
 PhD                   413
Name: Education, dtype: int64
```

In [22]:
```
df["Marital_status"]=df["Marital_status"].replace([' Married-civ-spouse',' Marr
df["Marital_status"]=df["Marital_status"].replace([' Never-married'],' Single')
df["Marital_status"]=df["Marital_status"].replace([' Divorced',' Widowed',' Sep
```

In [23]:
```
df["Marital_status"].value_counts()
```

Out[23]:
```
 Married    14999
 Single     10682
 Others      6879
Name: Marital_status, dtype: int64
```

- Now we have replaced the unique types classes in the columns Workclass, Education and Marital_status.

In [24]:
```
# let's check the dataframe
df.head()
```

Out[24]:

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relatio
0	50	Pvt-Sector	83311	Under_Graduation	13	Married	Exec-managerial	Hu
1	38	Pvt-Sector	215646	High-School	9	Others	Handlers-cleaners	Not-in-
2	53	Pvt-Sector	234721	School	7	Married	Handlers-cleaners	Hu
3	28	Pvt-Sector	338409	Under_Graduation	13	Married	Prof-specialty	
4	37	Pvt-Sector	284582	Post_Graduation	14	Married	Exec-managerial	

In [25]:
```
# checking the list of value counts in Income
df['Income'].value_counts()
```

Out[25]:
```
 <=50K    24719
 >50K      7841
Name: Income, dtype: int64
```

- There are two unique values in the target columns <=50k and >50k.

We can say that whether the person has annual income <=50k or >50k

We can also observe that the class imbalancing issue here so will balance the data using SMOTE before machine learning modeling.

In [26]:
```
# checking wheather the dataset contains any space
df.loc[df['Income']==" "]
```

Out[26]:

Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship	F

- It seems that there are no spaces in the dataset.

Description of Dataset

In [27]:
```python
# statistical summary of dataset
df.describe()
```

Out[27]:

	Age	Fnlwgt	Education_num	Hours_per_week
count	32560.000000	3.256000e+04	32560.000000	32560.000000
mean	38.581634	1.897818e+05	10.080590	40.437469
std	13.640642	1.055498e+05	2.572709	12.347618
min	17.000000	1.228500e+04	1.000000	1.000000
25%	28.000000	1.178315e+05	9.000000	40.000000
50%	37.000000	1.783630e+05	10.000000	40.000000
75%	48.000000	2.370545e+05	12.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99.000000

This gives the statistical information of the dataset . The summary of this dataset looks perfect since there is no negative/invalid values present.

From the above description we can observe the following things.

- The counts of all the columns are same which means there is no missing values present in any columns
- The mean is greater than the meadian(50%) in some columns which means they are skewed to right.
- The mean and the median(50%) are almost equal in Education_num and Hours_per_week which means the data is symmetric in these columns hence the data is normal and no skewness present here.
- There is a huge difference in 75% and max it shows that huge outliers present in the columns.
- In summarising the data we can observe that the dataset contains the person's age between 17 years to 90 years.

Let's Separate categorical and numerical columns

In [28]:
```python
# checking for categorical columns
categorical_col=[]
for i in df.dtypes.index:
    if df.dtypes[i]=='object':
        categorical_col.append(i)
print(categorical_col)
```

```
['Workclass', 'Education', 'Marital_status', 'Occupation', 'Relationship', 'R
ace', 'Sex', 'Native_country', 'Income']
```

- These are the categorical columns present in the dataset

In [29]:
```
# Now checking for numerical columns
numerical_col=[]
for i in df.dtypes.index:
    if df.dtypes[i]!='object':
        numerical_col.append(i)
print(numerical_col)
```

```
['Age', 'Fnlwgt', 'Education_num', 'Hours_per_week']
```

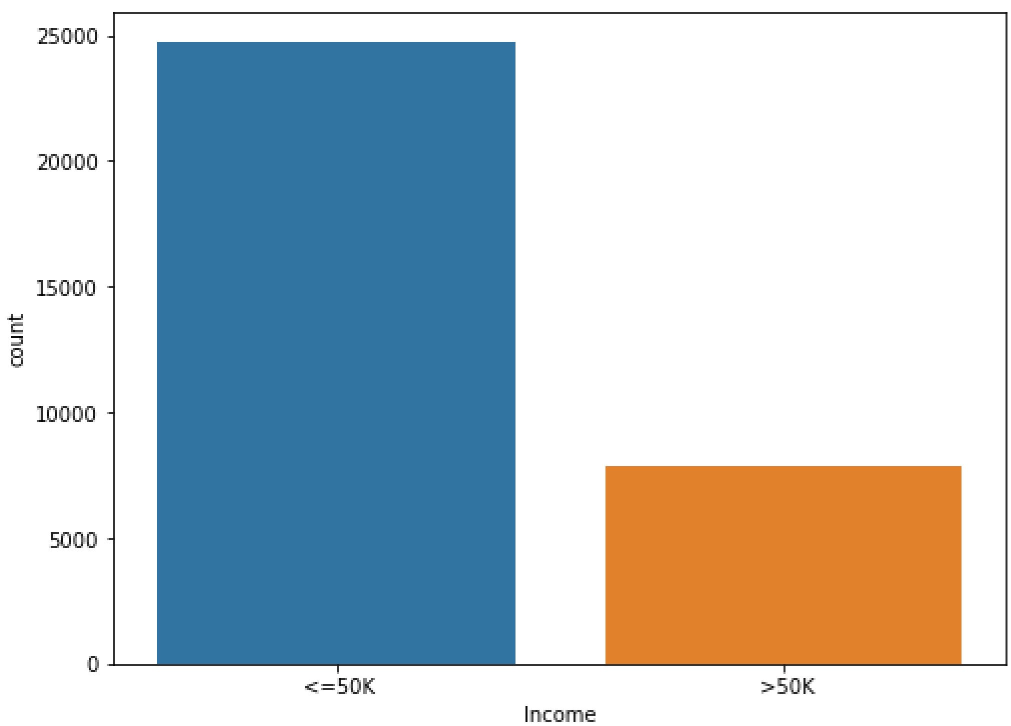
- These are the columns having numerical values

Data Visualization

Univariate Analysis

Plotting categorical columns

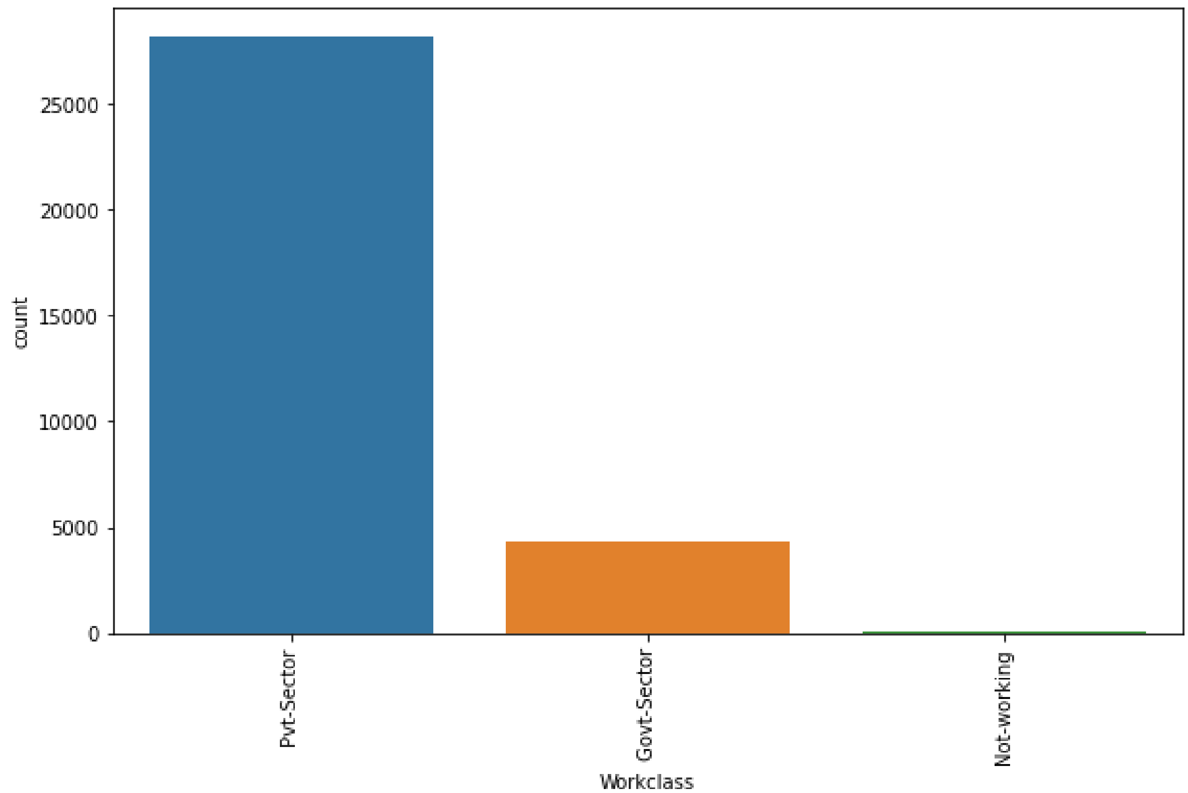
In [30]:
```
# Visualize the whether the income is above 50k or not
plt.figure(figsize=(8,6))
sns.countplot(df['Income'])
plt.show()
```



- Most of the people have the income less than or equal to 50k .

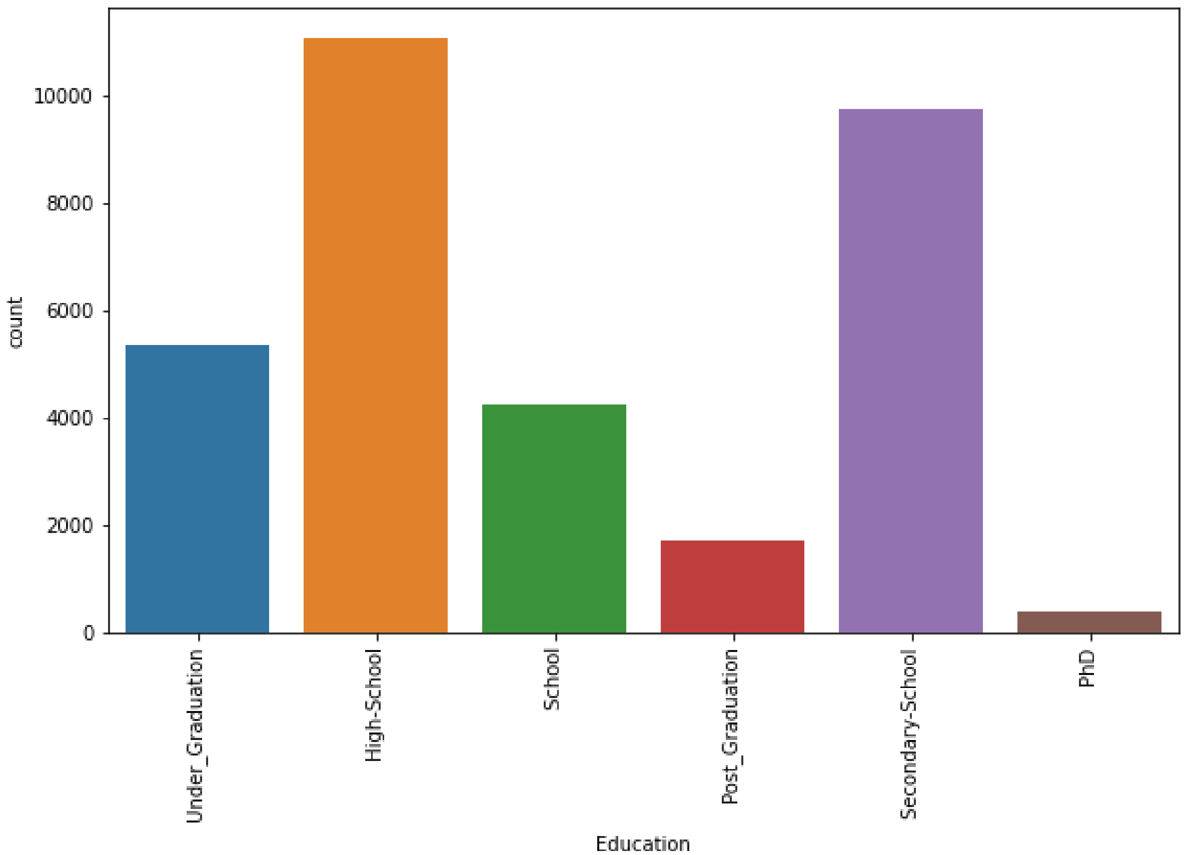
We can also observe the class imbalance so will balance the data before building our model.

In [31]:
```
# visualize the count of workclass of the people
plt.figure(figsize=(10,6))
sns.countplot(df['Workclass'])
plt.xticks(rotation=90)
plt.show()
```



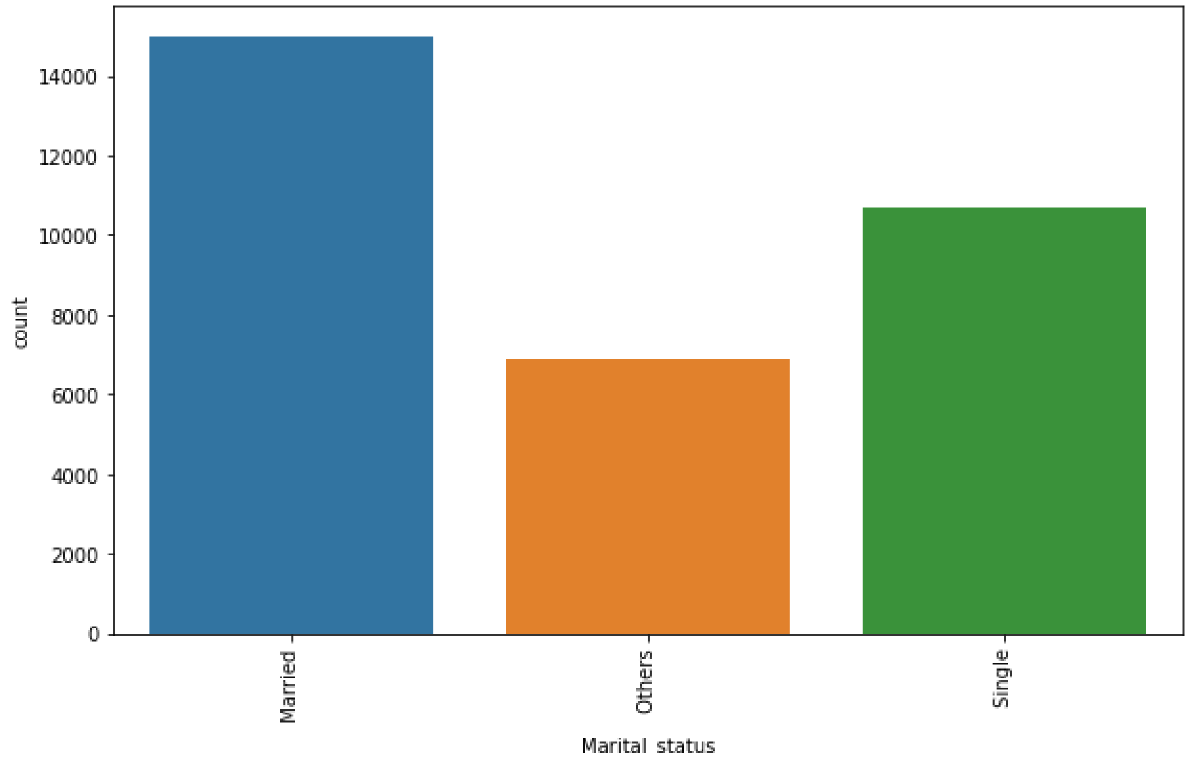
- The count of Private work class is high compare to others.
- This means the people working in private sectprs are high count and the people who never worked have least count.

In [32]:
```
# visualize the count Education of the people
plt.figure(figsize=(10,6))
sns.countplot(df['Education'])
plt.xticks(rotation=90)
plt.show()
```



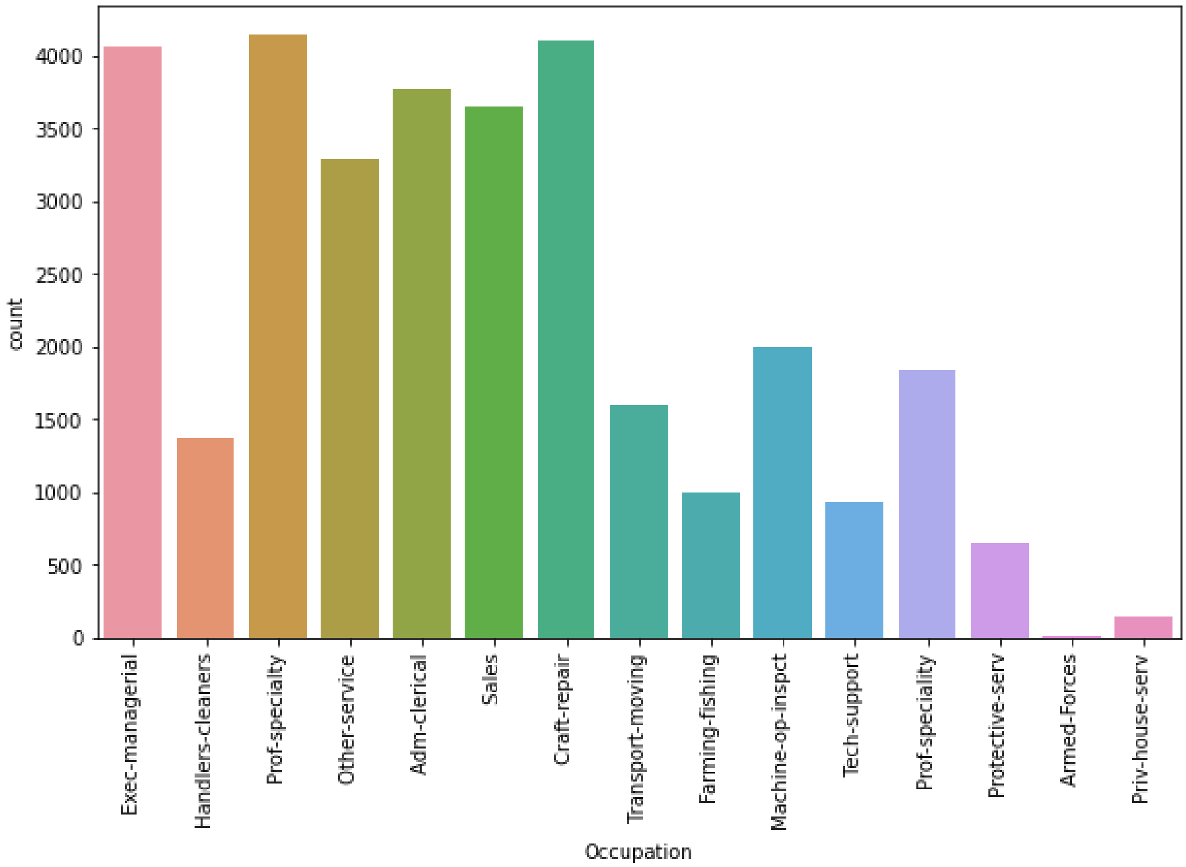
- The count of High-School is high followed by Secondary_School.
- Most of the people have their high school graduation with count more than 10k and the count of PhD is very less comapare to others.

In [33]:
```
# visualize the marital status of the people
plt.figure(figsize=(10,6))
sns.countplot(df["Marital_status"])
plt.xticks(rotation=90)
plt.show()
```



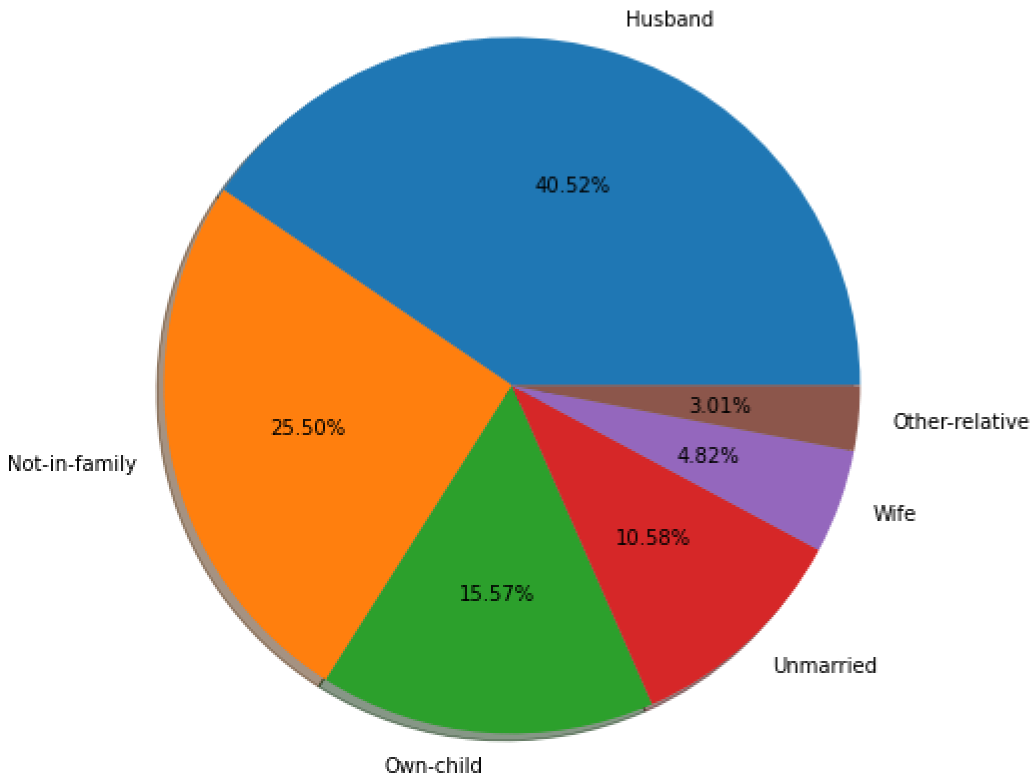
- The people who got married have high count followed by the singles and never married people

In [34]:
```
# visualize the count of Occupation of the people
plt.figure(figsize=(10,6))
sns.countplot(df["Occupation"])
plt.xticks(rotation=90)
plt.show()
```



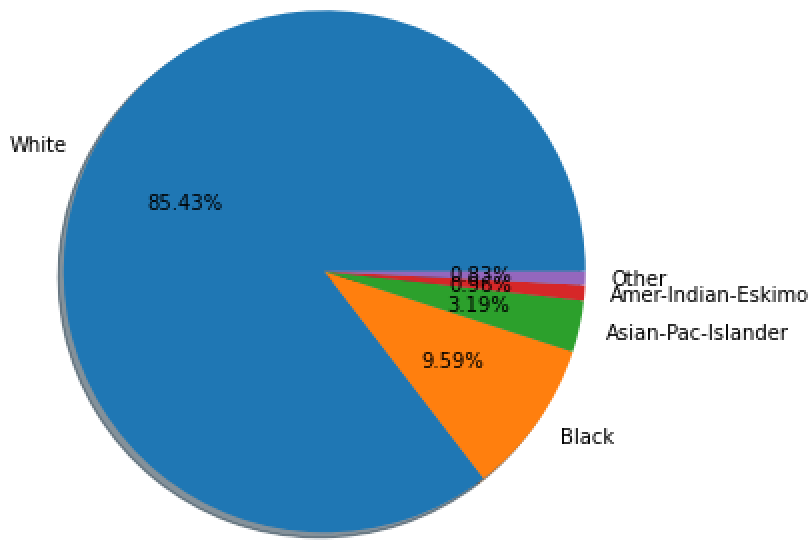
- The peole who are in the position of Prof-speciality have highest count and the people in the position Armed-Forces have very least counts.

```
In [35]: # visualize the count of Relationship of the people
labels = 'Husband', 'Not-in-family','Own-child','Unmarried','Wife','Other-relat
fig, ax = plt.subplots(figsize=(10,8))
ax.pie(df['Relationship'].value_counts(), labels=labels, autopct='%1.2f%%', sha
plt.show()
```



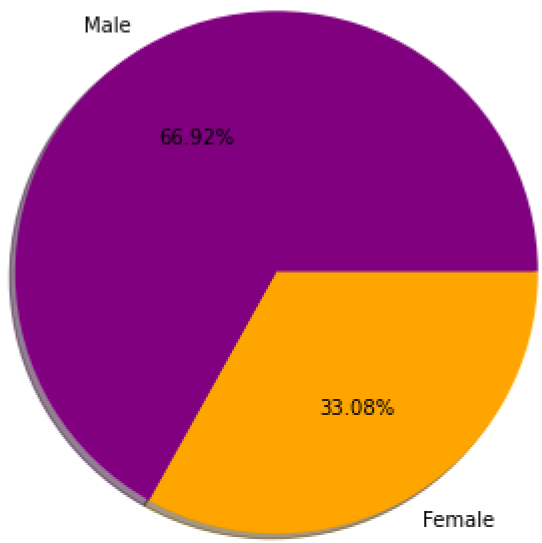
- The count is high in the Husband category which has around 40% of count and other relative has very least count around 3%

In [36]:
```
# visualize the count of Race of the people

labels='White','Black','Asian-Pac-Islander','Amer-Indian-Eskimo','Other'
fig, ax = plt.subplots(figsize=(10,6))
ax.pie(df['Race'].value_counts(), labels=labels, autopct='%1.2f%%', shadow=True
plt.show()
```



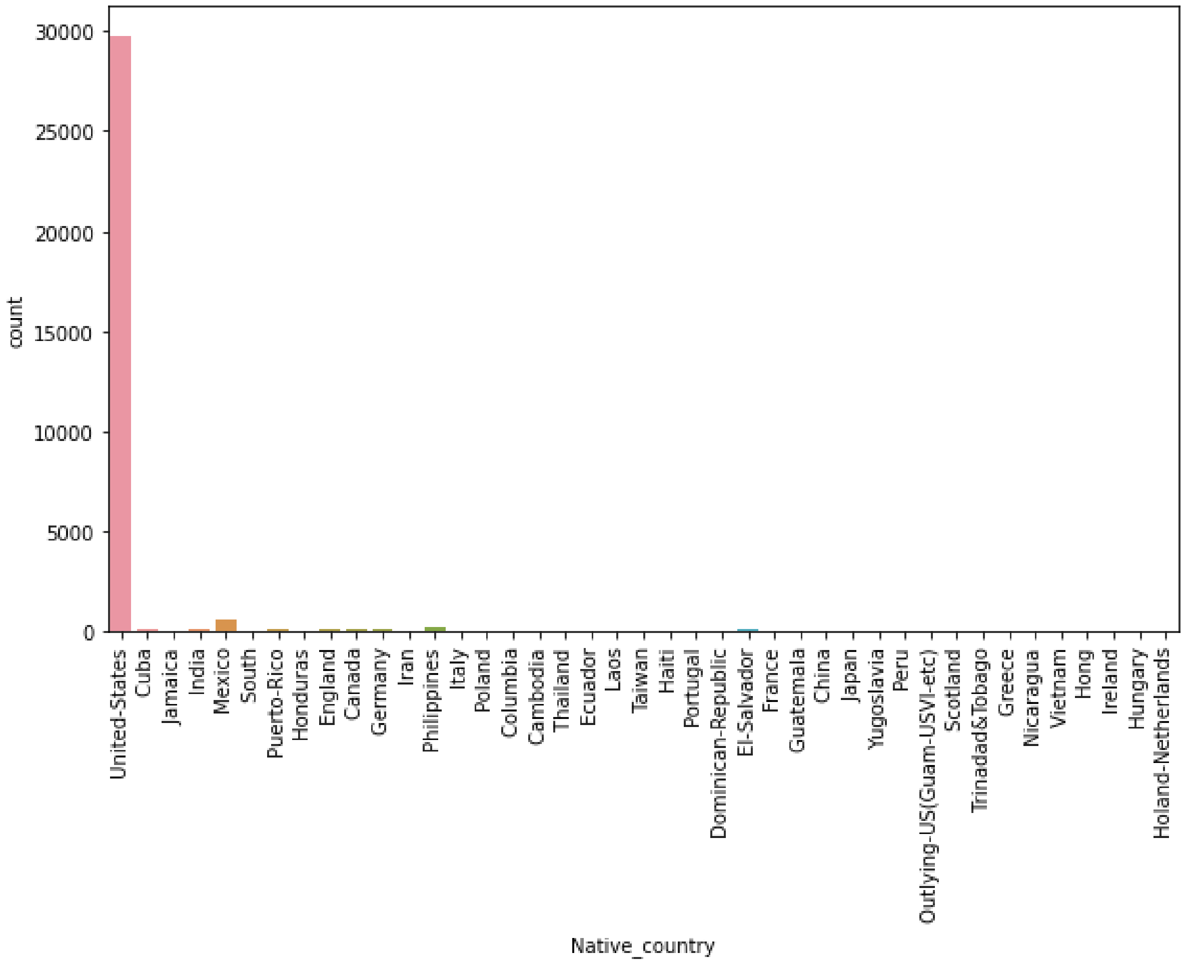
- White family group have high count of around 85% and Other have least count around 0.83%.

In [37]:
```
# visualize the count of Sex group of the people

labels='Male','Female'
fig, ax = plt.subplots(figsize=(10,6))
colors = ["purple","orange"]
ax.pie(df['Sex'].value_counts(), labels=labels, autopct='%1.2f%%', shadow=True,
plt.show()
```



- The count of Male is high and has around 66% and only 33% of females are there.

In [38]:
```
# visualizing the Native country of the people
plt.figure(figsize=(10,6))
sns.countplot(df['Native_country'])
plt.xticks(rotation=90)
plt.show()
```

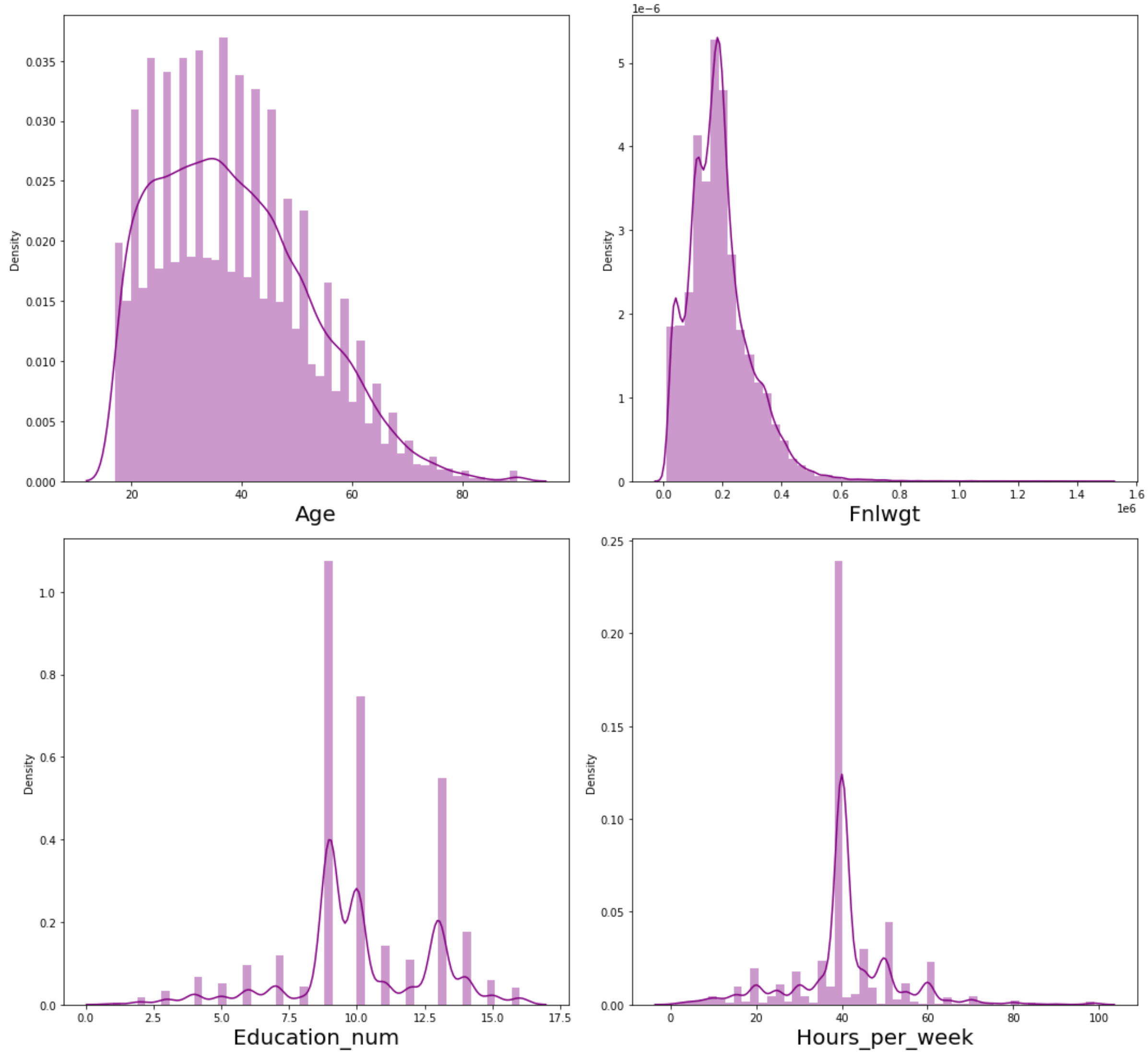


- The United States country has highest count of around 29k and other countries have very less counts.

Distribution of skewness

Plotting numerical columns

In [39]:
```
# checking how the data has been distributed in each column

plt.figure(figsize=(15,20),facecolor='white')
plotnumber=1
for column in numerical_col:
    if plotnumber<=6:
        ax=plt.subplot(3,2,plotnumber)
        sns.distplot(df[column], color='purple')
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```

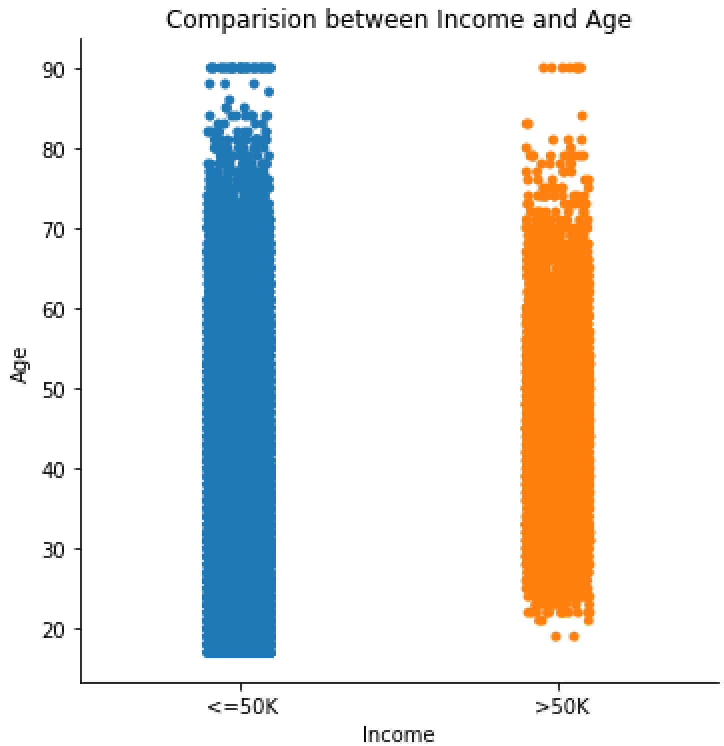


- From the above distribution plot it can be inferred that Age column seems to be normal but the mean is more than the median , so it is skewed to right.
- The data is not normal in the above columns and the columns final weight, capital gain an dcapital loss have right skewness since the mean is more in this case.
- The data in the columns Education num and Hoursper week are not normal but they have no skewness.

Bivariate Analysis

Age

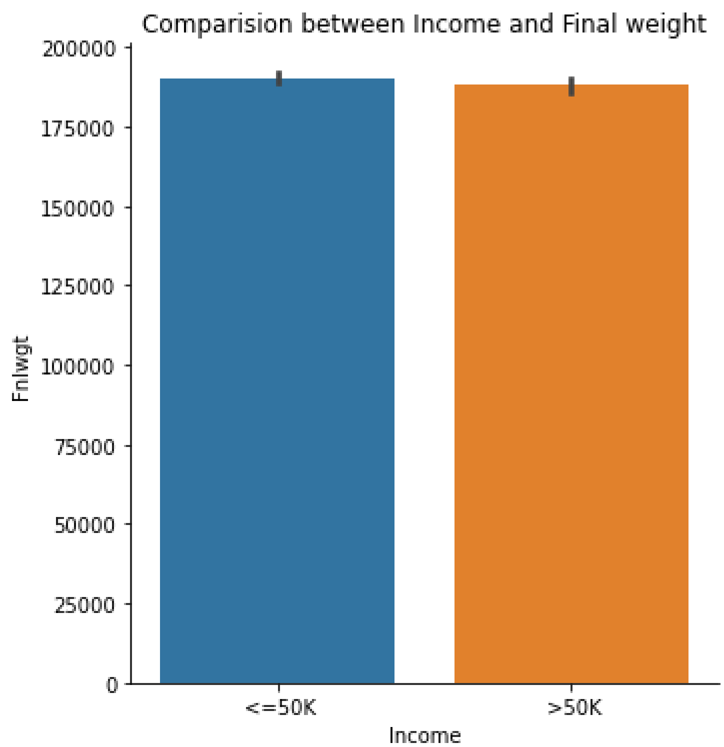
In [40]:
```
# visualizing the age of the people who have the income more
sns.catplot(df['Income'],df['Age'], data=df,kind='strip',size=5);
plt.title('Comparision between Income and Age')
plt.show()
```



- The people whose age is between 20 to 80 have annual income more than 50k

Final Weight

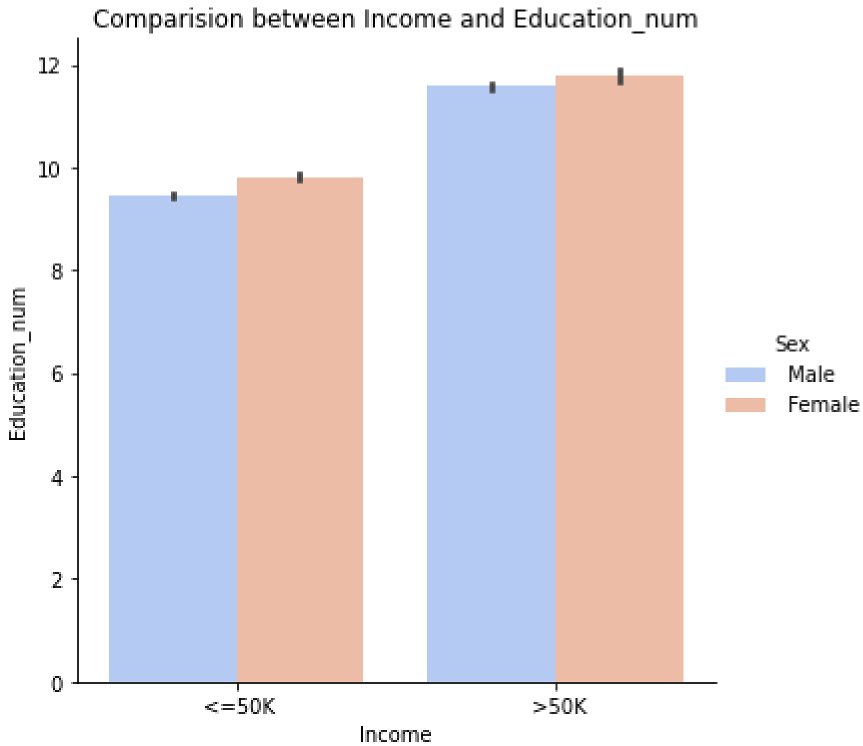
In [41]:
```
# visualizing the Final weight with income
sns.catplot(df['Income'],df['Fnlwgt'], data=df,kind='bar');
plt.title('Comparision between Income and Final weight')
plt.show()
```



- There is no significant relation between final weight and income of the people

Education_num

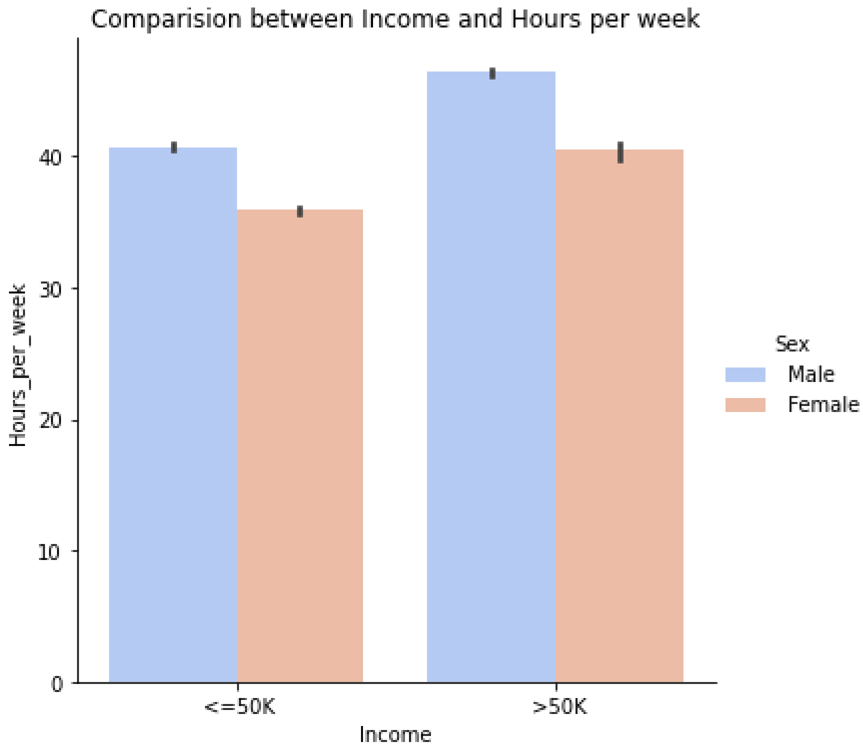
In [42]:
```
# visualizing the number of education with income
sns.catplot(x='Income',y='Education_num', data=df,kind='bar',hue="Sex",palette=
plt.title("Comparision between Income and Education_num")
plt.show()
```



- The Income is more than 50k for the people having high education number . Here both gender have the income more than 50k

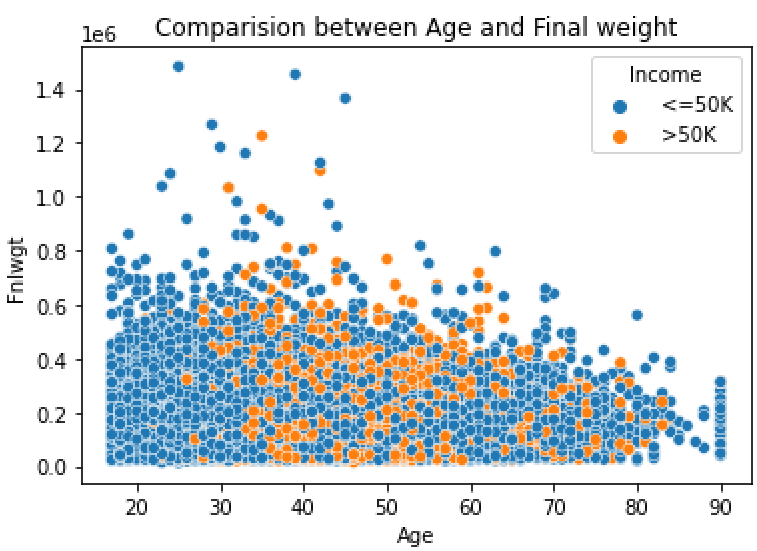
Hour Per Week

In [43]:
```
# visualizing the number of Hours per week with income
sns.catplot(x='Income',y='Hours_per_week', data=df,kind='bar', hue='Sex',palett
plt.title('Comparision between Income and Hours per week')
plt.show()
```



- This shows how the income is related to the hours per week . The income is >50k when the Hours is high for both male and female.

In [44]:
```
# visualizing how the income changes with work class of the people
sns.scatterplot(x='Age',y='Fnlwgt', data=df,hue='Income');
plt.title('Comparision between Age and Final weight')
plt.show()
```

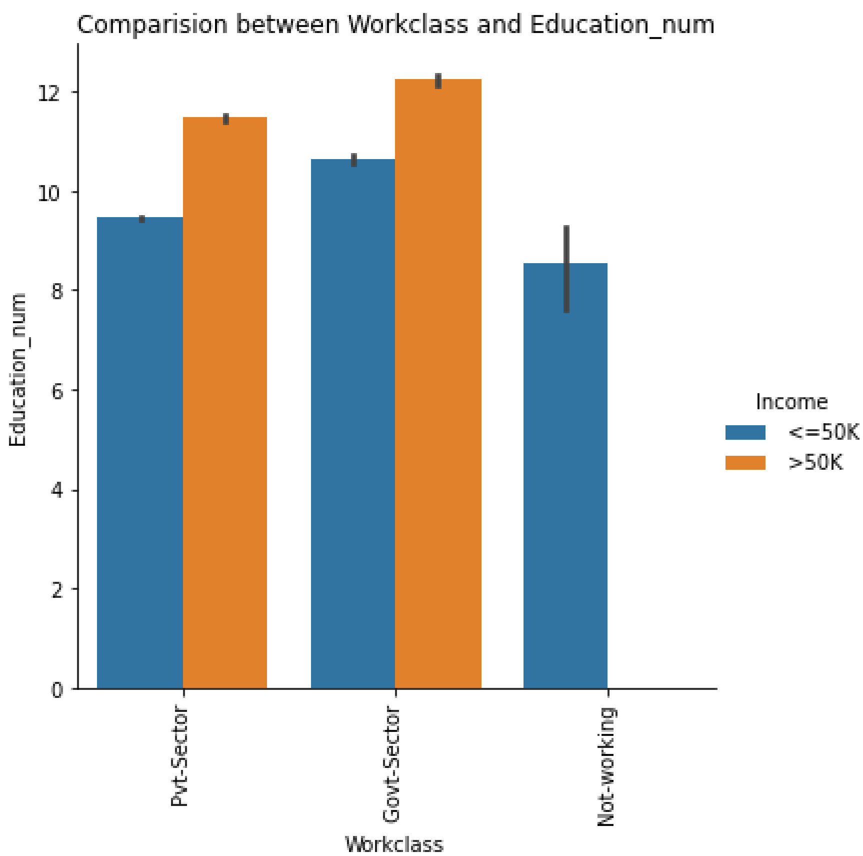


- The people's age between 17-80 with average final weight have income <=50k

Workclass

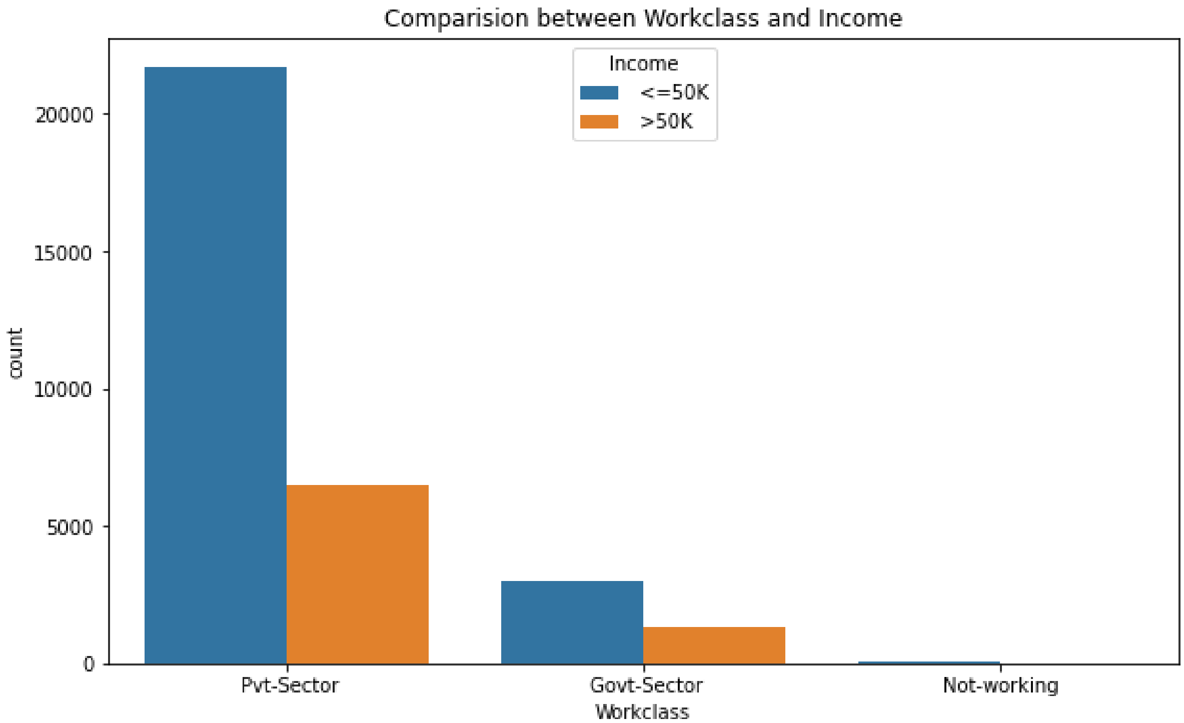
In [45]:
```
# visualizing how the income changes with work class of the people
plt.figure(figsize=(10,6))
sns.catplot(x='Workclass',y='Education_num',data=df,kind='bar',hue='Income');
plt.title('Comparision between Workclass and Education_num')
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```



- The people in the position of government jobs with high education number have the income >50k also the people in the Private sector position with average education number have second highest income >50k

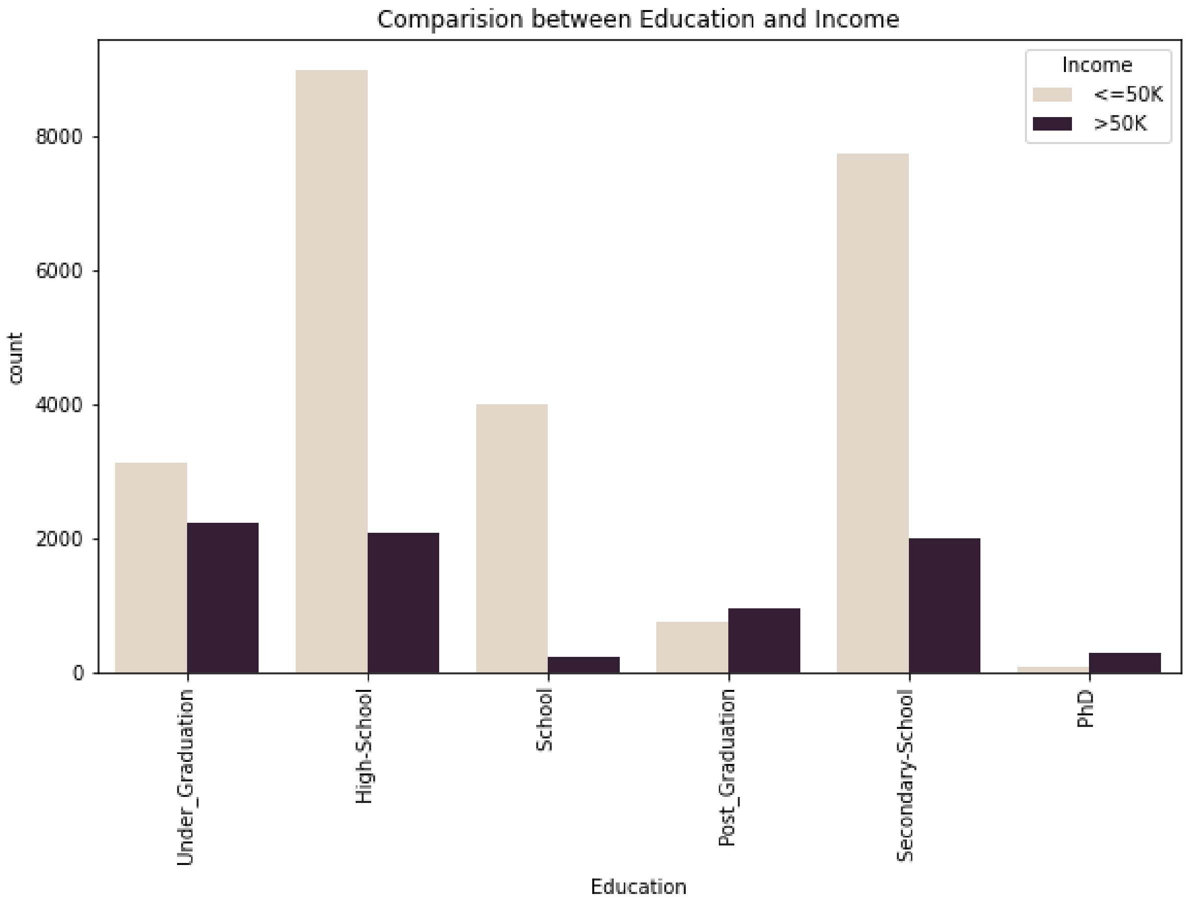
In [46]:
```
# visualizing the relation between work class and Income of the people
plt.figure(figsize=(10,6))
sns.countplot(df['Workclass'],hue=df['Income'])
plt.title('Comparision between Workclass and Income')
plt.show()
```



- The people who are working in the private sectors have the income <=50k an dthe only few of the people in the same sector have income >50k .
- Also the people who never worked they don't have the income.

Education

In [47]:
```
# visualizing the relation between Education and Income of the people
plt.figure(figsize=(10,6))
sns.countplot(df['Education'],hue=df['Income'],palette="ch:.25")
plt.title('Comparision between Education and Income')
plt.xticks(rotation=90)
plt.show()
```

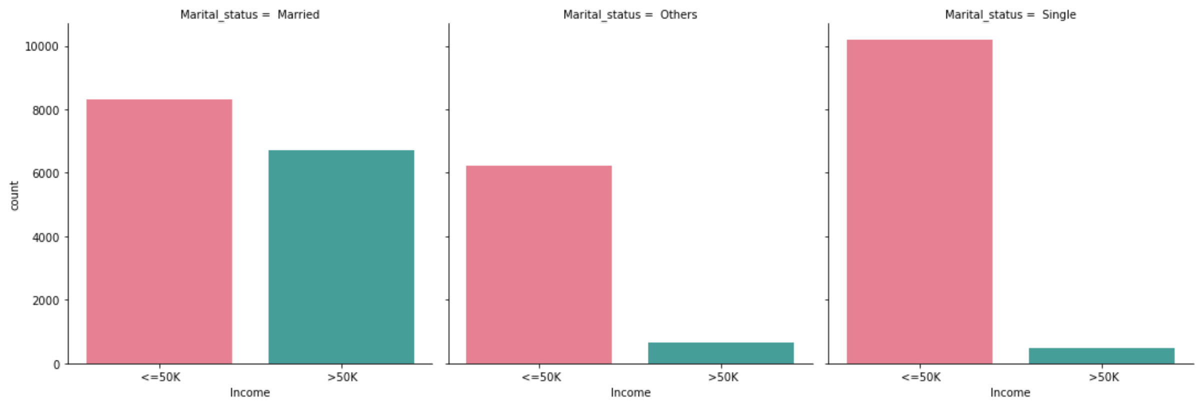


- The people who completed there high school have income <=50k followed by the people who done their Secondary School .
- Also the people who done their Graduatuion they are earning more income that is >50k

Marital_status

In [48]:
```
# visualizing the relation between Marital status and Income of the people
plt.figure(figsize=(10,6))
sns.catplot(x='Income', col='Marital_status',data=df,kind='count',palette='husl
plt.show()
```

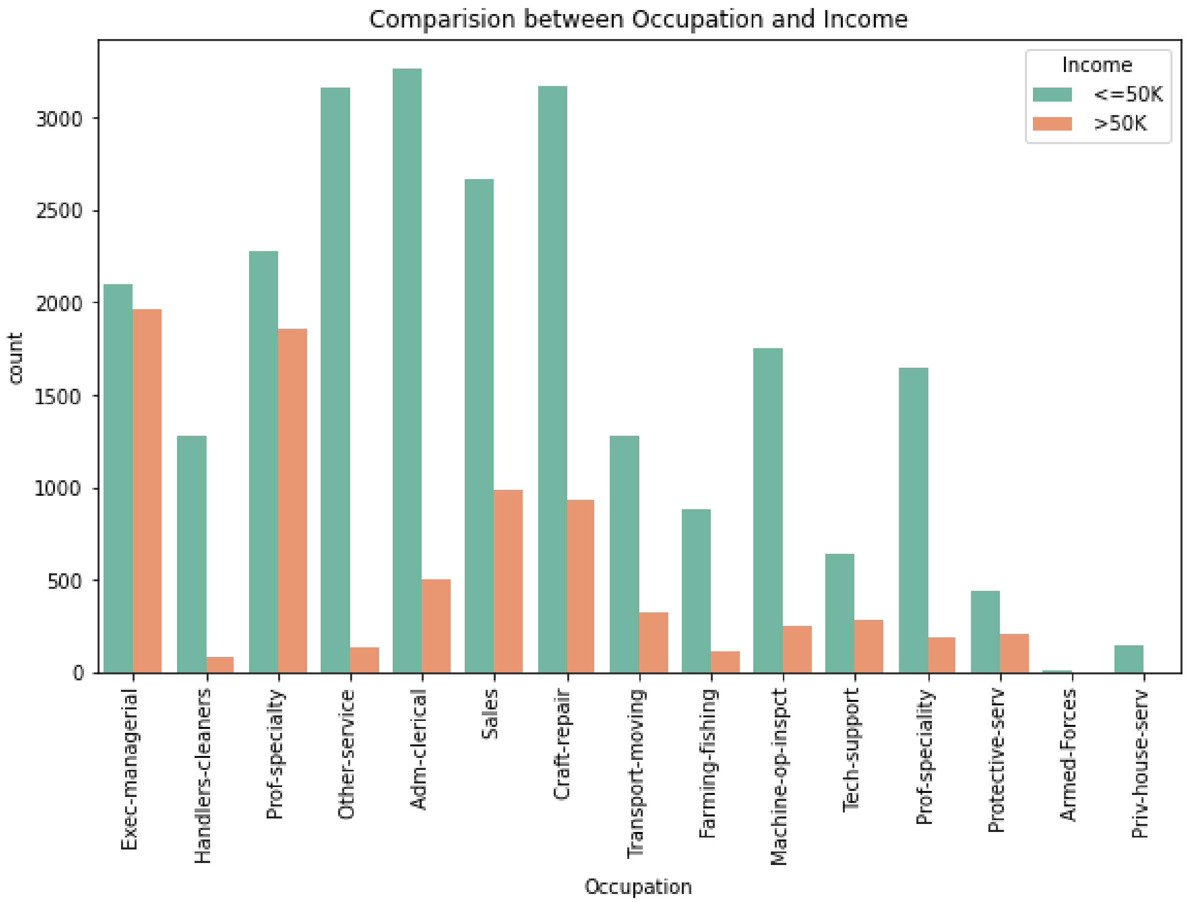
```
<Figure size 720x432 with 0 Axes>
```



- The people who are married they have the income >50k compare to others.
- The people who are staying singles earning <=50k income.

Occupation

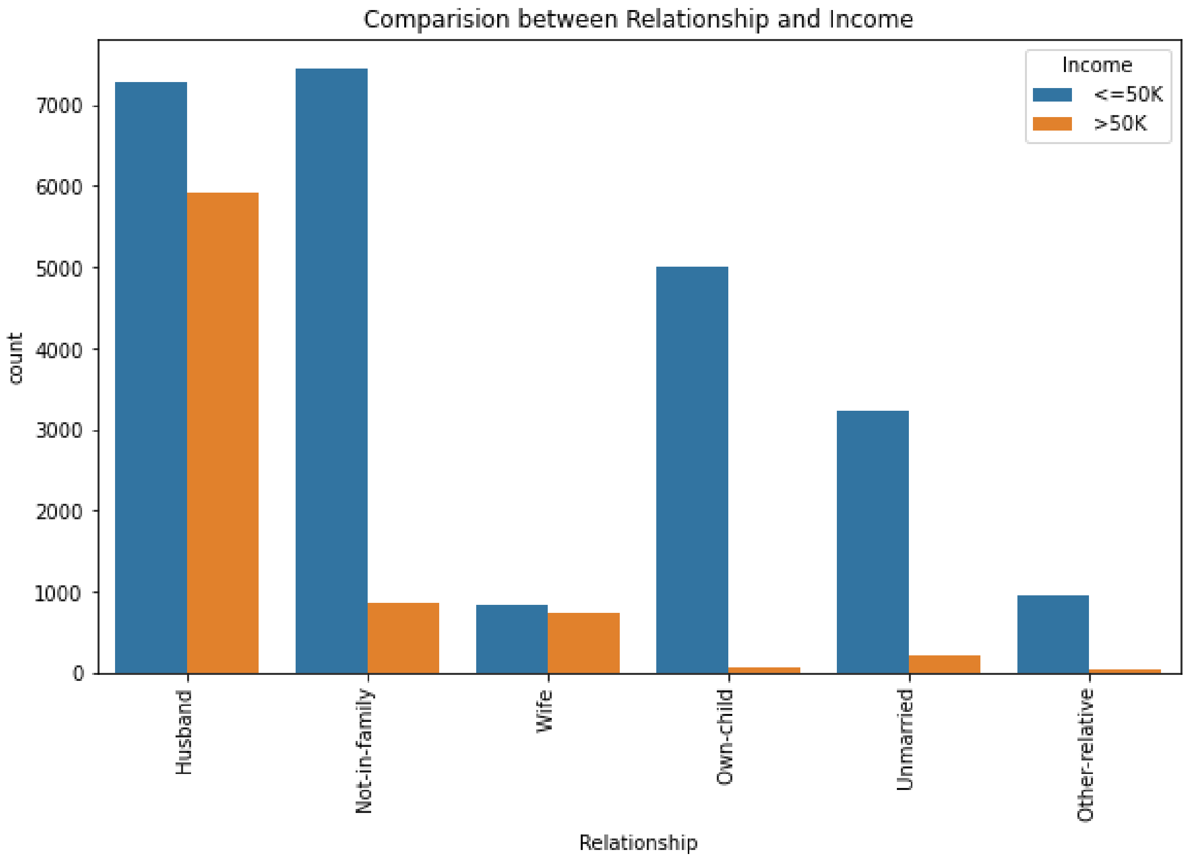
In [49]:
```
# visualizing the relation between Occupation and Income of the people
plt.figure(figsize=(10,6))
sns.countplot(df['Occupation'],hue=df['Income'],palette='Set2')
plt.title("Comparision between Occupation and Income")
plt.xticks(rotation=90)
plt.show()
```



- The people in the position Prof-speciality and Exce-managerial have the income more than 50k
- Also the people who are in the position Prof_Speciality, Other sevices, Adm-clerical and craft repair they have income less than 50k

Relationship

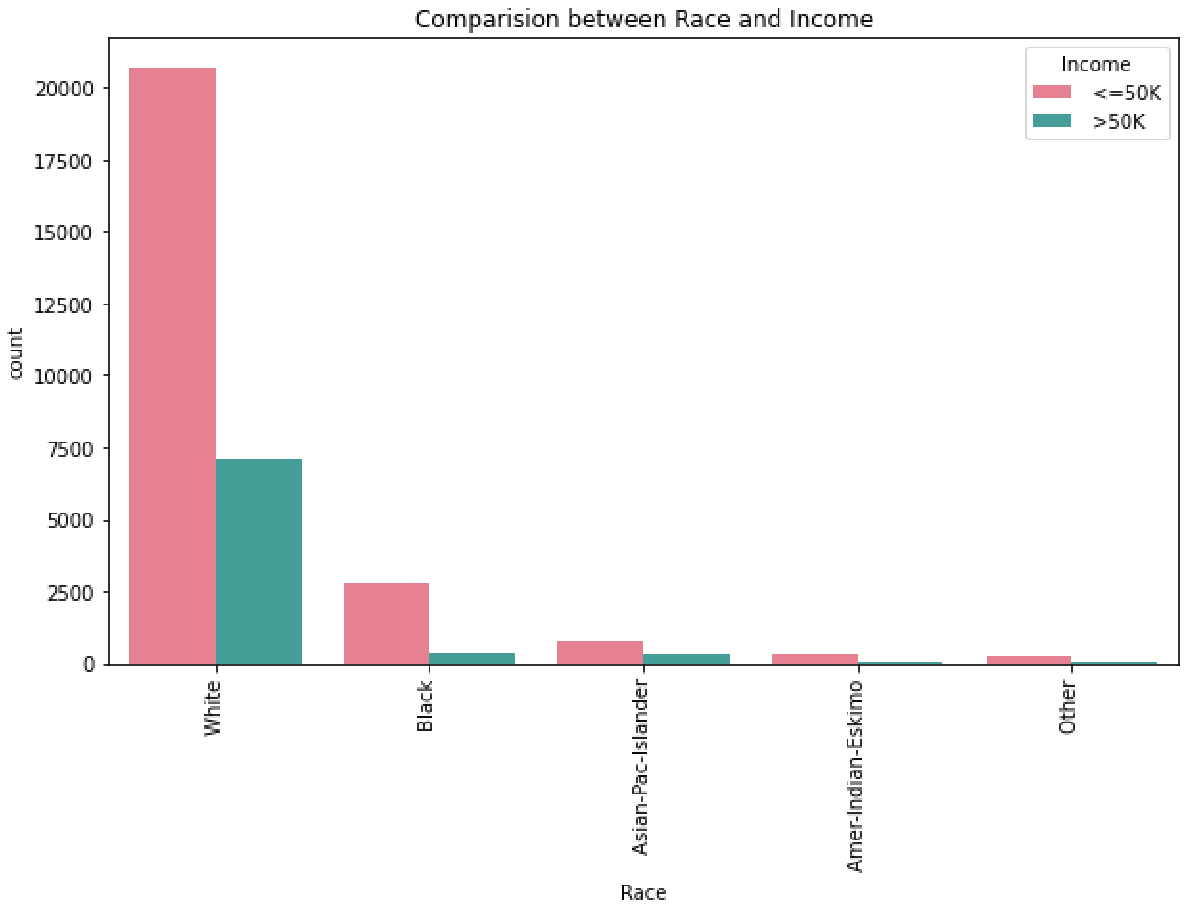
In [50]:
```
# visualizing the relation between Relationship and Income of the people
plt.figure(figsize=(10,6))
sns.countplot(df['Relationship'],hue=df['Income'])
plt.title('Comparision between Relationship and Income')
plt.xticks(rotation=90)
plt.show()
```



- People who have the relationship of husband and wife have income >50k and the others relationship giving income <=50k

Race

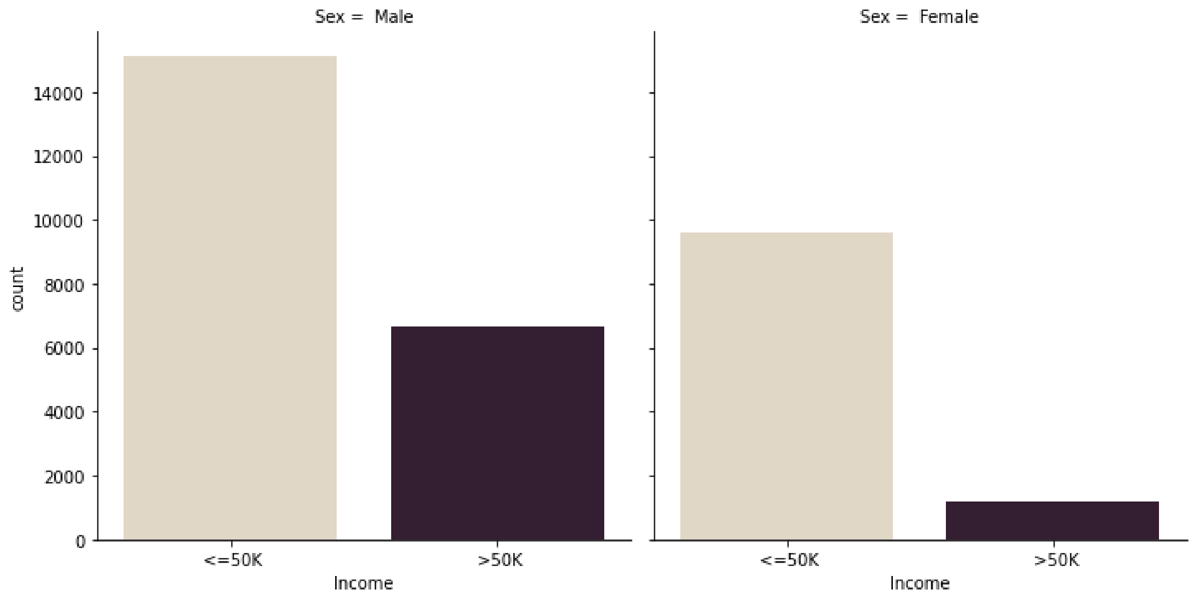
In [51]:
```
# visualizing the relation between Race and Income of the people
plt.figure(figsize=(10,6))
sns.countplot(df['Race'],hue=df['Income'],data=df,palette="husl")
plt.title('Comparision between Race and Income')
plt.xticks(rotation=90)
plt.show()
```



- The White family groups have high income >50k compare to other groups

Sex

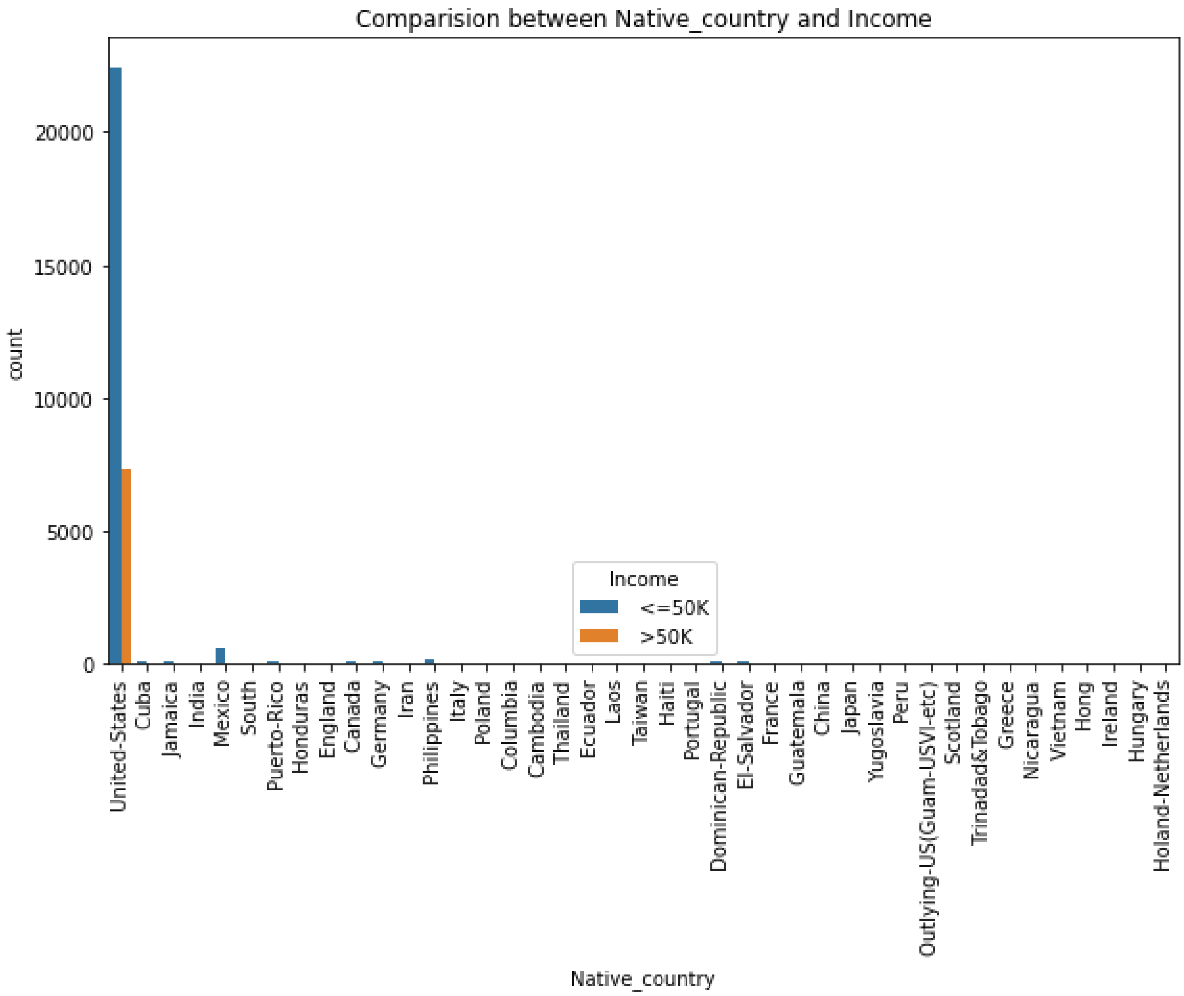
In [52]:
```
# visualizing the relation between Income and Sex groups of the people
sns.catplot(x='Income',col='Sex',data=df,kind='count',palette="ch:.28")
plt.show()
```



- The income of Male is above 50k compared to the female.

Native country

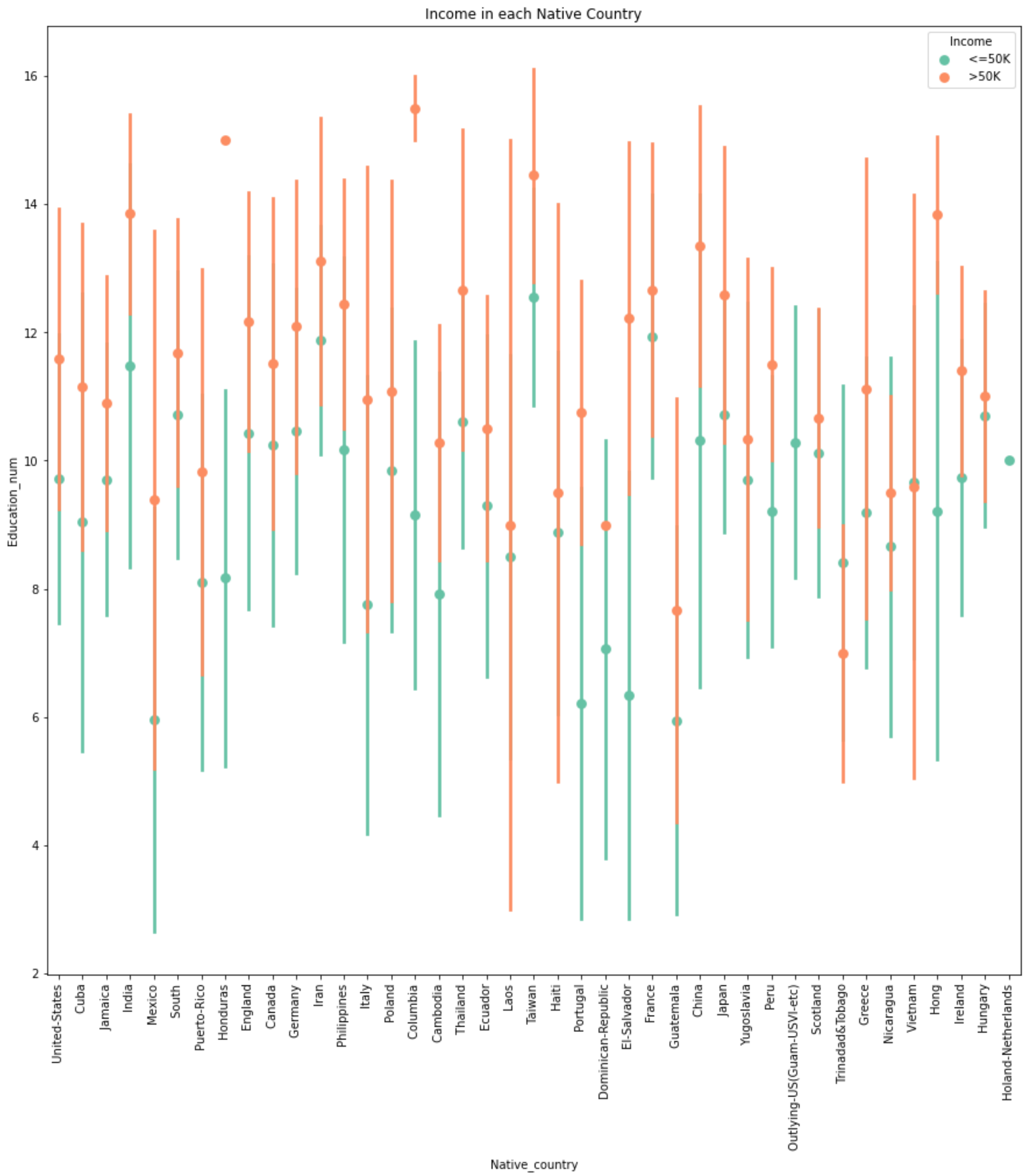
In [53]:
```
# visualizing the relation between Native country and Income of the people
plt.figure(figsize=(10,6))
sns.countplot(df['Native_country'],hue=df['Income'])
plt.title('Comparision between Native_country and Income')
plt.xticks(rotation=90)
plt.show()
```



- United States earning more income compared to all the other countries.

In [54]:

```
# visualizing how the income changes for Native country of the people
plt.figure(figsize=(15,15))
plt.title('Income in each Native Country')
sns.pointplot(x='Native_country',y='Education_num',data=df, hue='Income',join=F
plt.xticks(rotation=90)
plt.show()
```



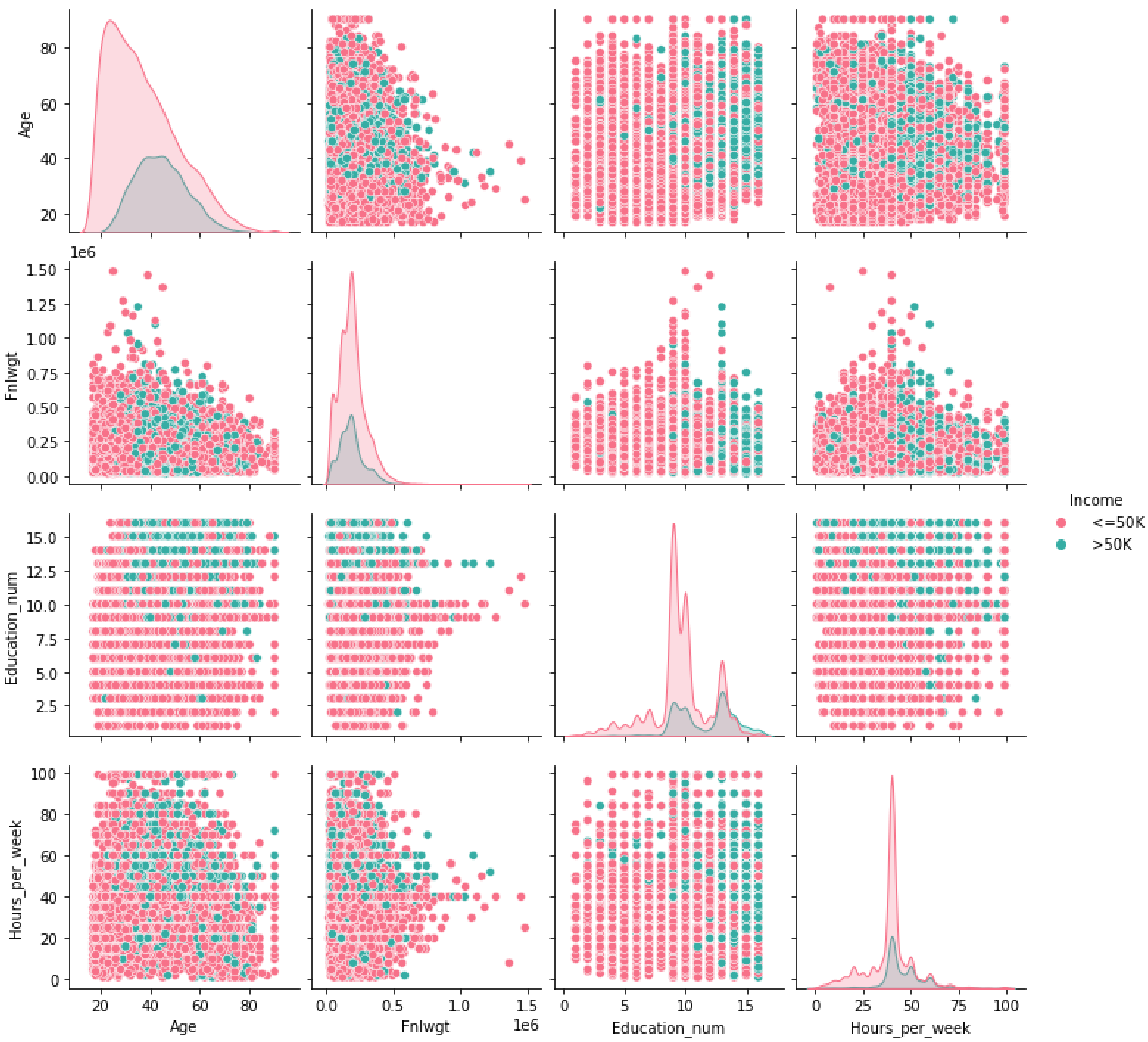
- The countries having high education numbers have high income that is more than 50k

In [55]:
```python
# visualizing the relationship between Sex and Age of the people

sns.catplot(x='Sex',y='Age', kind='violin',color='g',data=df,hue='Income',split
plt.title("Comparision between Sex and Age")
plt.show()
```

- The income of male with age 17-55 have the income >50k compared to the female

In [56]:
```
# checking the pairwise relation in the dataset.
sns.pairplot(df,hue='Income',palette='husl')
```

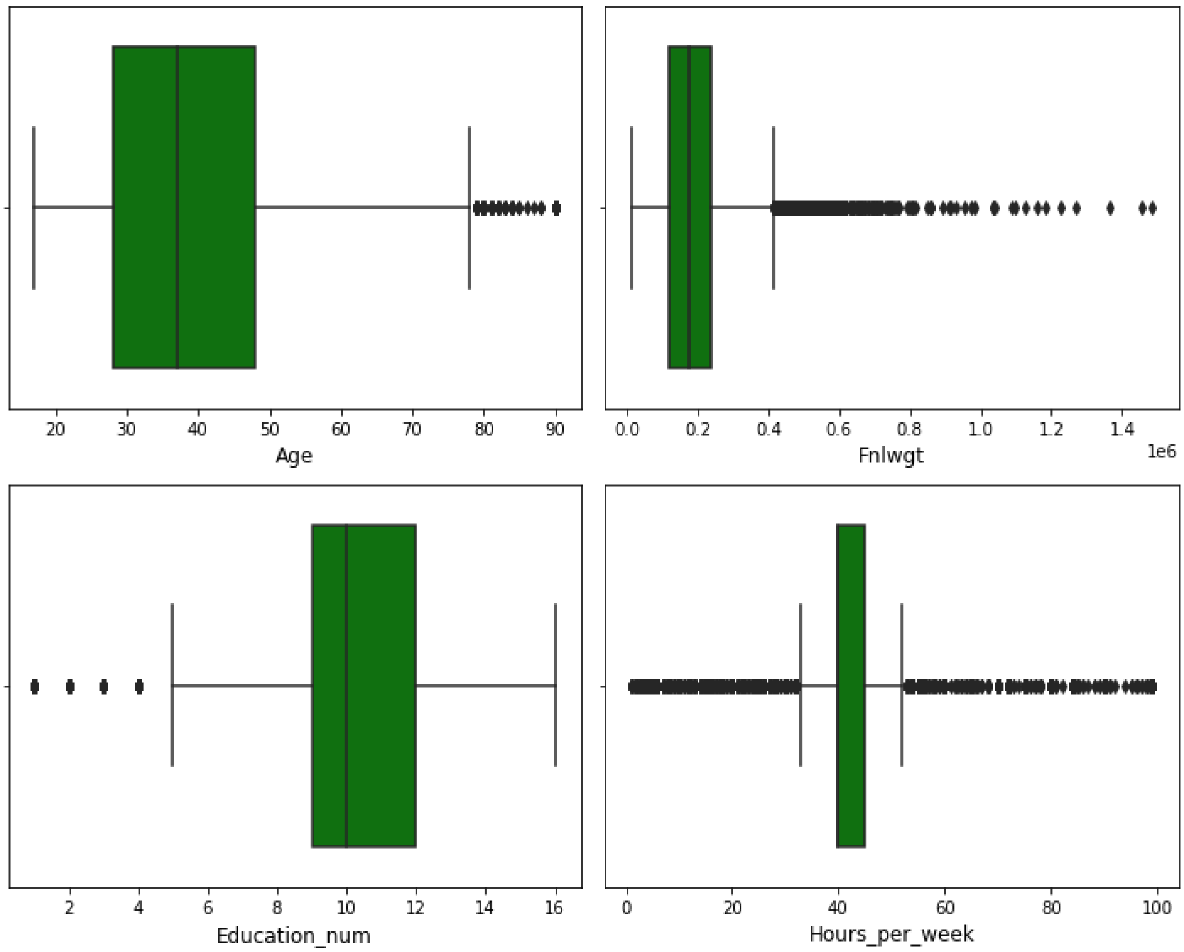
Out[56]: <seaborn.axisgrid.PairGrid at 0x190c563e280>



- This pair plot gives the pairwise relation between the columns which is plotted on the basis of target variable 'Income'. Here we ca observe the relation between the features and label.
- Most of features are highly correlated with each other.
- Some of th efeatures have outliers and skewness , will remove them later.

Outliers Handling

In [57]:
```
plt.figure(figsize=(10,8),facecolor='white')
plotnumber=1
for column in numerical_col:
    if plotnumber<=4:
        ax=plt.subplot(2,2,plotnumber)
        sns.boxplot(df[column],color='g')
        plt.xlabel(column,fontsize=12)
    plotnumber+=1
plt.tight_layout()
```



- The outliers present in all the columns we will remove it using ZSCORE method.

Removing Outliers

1. ZSCORE Method

In [58]:
```
features = df[['Age','Fnlwgt','Education_num','Hours_per_week']]
from scipy.stats import zscore
z=np.abs(zscore(features))
z
```

Out[58]:
```
array([[0.83709708, 1.0087417 , 1.13477863, 2.22212013],
       [0.04264043, 0.24504633, 0.42002663, 0.03542999],
       [1.05703146, 0.42576955, 1.19742926, 0.03542999],
       ...,
       [1.42358875, 0.3588108 , 0.42002663, 0.03542999],
       [1.21562378, 0.11092744, 0.42002663, 1.65520046],
       [0.98372   , 0.92986178, 0.42002663, 0.03542999]])
```

- Now we have removed the outliers using ZSCORE method

In [59]:
```
# creating new dataframe
new_df = df[(z<3).all(axis=1)]
new_df
```

Out[59]:

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Re
0	50	Pvt-Sector	83311	Under_Graduation	13	Married	Exec-managerial	
1	38	Pvt-Sector	215646	High-School	9	Others	Handlers-cleaners	N
2	53	Pvt-Sector	234721	School	7	Married	Handlers-cleaners	
3	28	Pvt-Sector	338409	Under_Graduation	13	Married	Prof-specialty	
4	37	Pvt-Sector	284582	Post_Graduation	14	Married	Exec-managerial	
...	...	...	...	...	...	...	...	
32555	27	Pvt-Sector	257302	Secondary-School	12	Married	Tech-support	
32556	40	Pvt-Sector	154374	High-School	9	Married	Machine-op-inspct	
32557	58	Pvt-Sector	151910	High-School	9	Others	Adm-clerical	
32558	22	Pvt-Sector	201490	High-School	9	Single	Adm-clerical	
32559	52	Pvt-Sector	287927	High-School	9	Married	Exec-managerial	

31461 rows × 13 columns

- This is the new dataframe after removing the outliers. Here we have removed the outliers whose zscore is less than 3.

In [60]:
```
# shape of original dataset
df.shape
```

Out[60]: (32560, 13)

- Before removing the outliers we had 32560 rows and 13 columns in our dataset.

In [61]:
```
# shape of new dataframe
new_df.shape
```

Out[61]: (31461, 13)

- After removing the outliers we had 31461 rows and 13 columns

In [62]:
```
# checking the data loss%
data_loss =(32560-31461)/32560*100
```

In [63]:
```
data_loss
```

Out[63]: 3.3753071253071254

- Here we are lossing onlu 3 % data using zscore method

2. IQR Method

In [64]:
```
Q1 = features.quantile(0.25)
Q3 = features.quantile(0.75)

IQR=Q3 - Q1

df1 = df[~((df <(Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

In [65]:
```
df1.shape
```

Out[65]: (21950, 13)

- Using IQR method the dataframe has 21950 rows and 13 columns

In [66]:
```
# checking the data loss %
data_loss = (32560-21950)/32560*100
data_loss
```

Out[66]: 32.58599508599509

- Using IQR method we are losing 32% of data,which is huge
- So considering ZSCORE METHOD

Checking the skewness

In [67]:
```
new_df.skew()
```

Out[67]:
```
Age               0.472279
Fnlwgt            0.634828
Education_num    -0.159752
Hours_per_week   -0.341724
dtype: float64
```

- We can find the skewness in the columns Fnlwgt, let's remove it using cube root method.

Removing Skewness

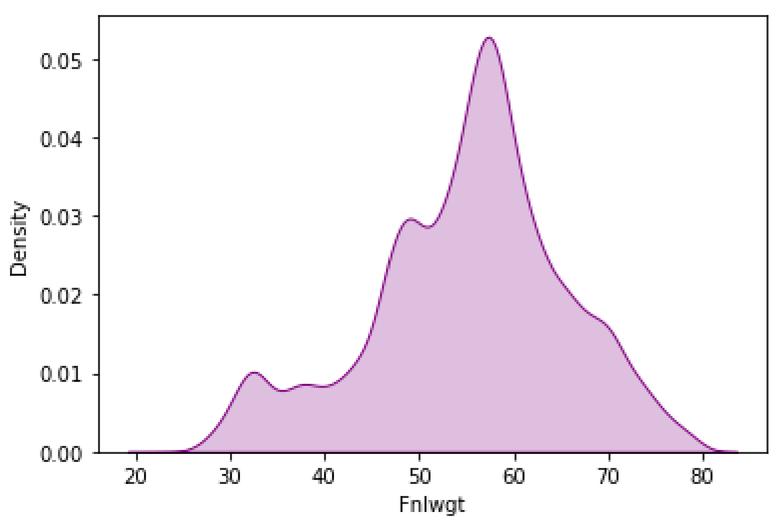
In [68]:
```
new_df['Fnlwgt'] = np.cbrt(df['Fnlwgt'])
```

In [69]:
```
new_df.skew()
```

Out[69]:
```
Age               0.472279
Fnlwgt           -0.376609
Education_num    -0.159752
Hours_per_week   -0.341724
dtype: float64
```

In [70]:
```
# After removing skewness. Let's check how the data has been distributed in eac
sns.distplot(new_df['Fnlwgt'],color='purple',kde_kws={'shade': True},hist=False
```

Out[70]:
```
<AxesSubplot:xlabel='Fnlwgt', ylabel='Density'>
```



- The data is almost normal and has no skewness.

Encoding the categorical columns using Label Encoding

In [71]:
```
categorical_col = ['Workclass','Education','Marital_status','Occupation','Relat

from sklearn.preprocessing import LabelEncoder
LE=LabelEncoder()
new_df[categorical_col]=new_df[categorical_col].apply(LE.fit_transform)
```

- Encoding the categorical columns using label encoder.

In [72]:
```
new_df[categorical_col]
```

Out[72]:

	Workclass	Education	Marital_status	Occupation	Relationship	Race	Sex	Native_country
0	1	5	0	3	0	4	1	38
1	1	0	1	5	1	4	1	38
2	1	3	0	5	0	2	1	38
3	1	5	0	10	5	2	0	4
4	1	2	0	3	5	4	0	38
...	...	...	...	...	...	...	...	..
32555	1	4	0	13	5	4	0	38
32556	1	0	0	6	0	4	1	38
32557	1	0	1	0	4	4	0	38
32558	1	0	2	0	3	4	1	38
32559	1	0	0	3	5	4	0	38

31461 rows × 9 columns

- Categorical columns after encoding the data using label encoding method.

Correlation between the target variable and independent variables using HEAT map

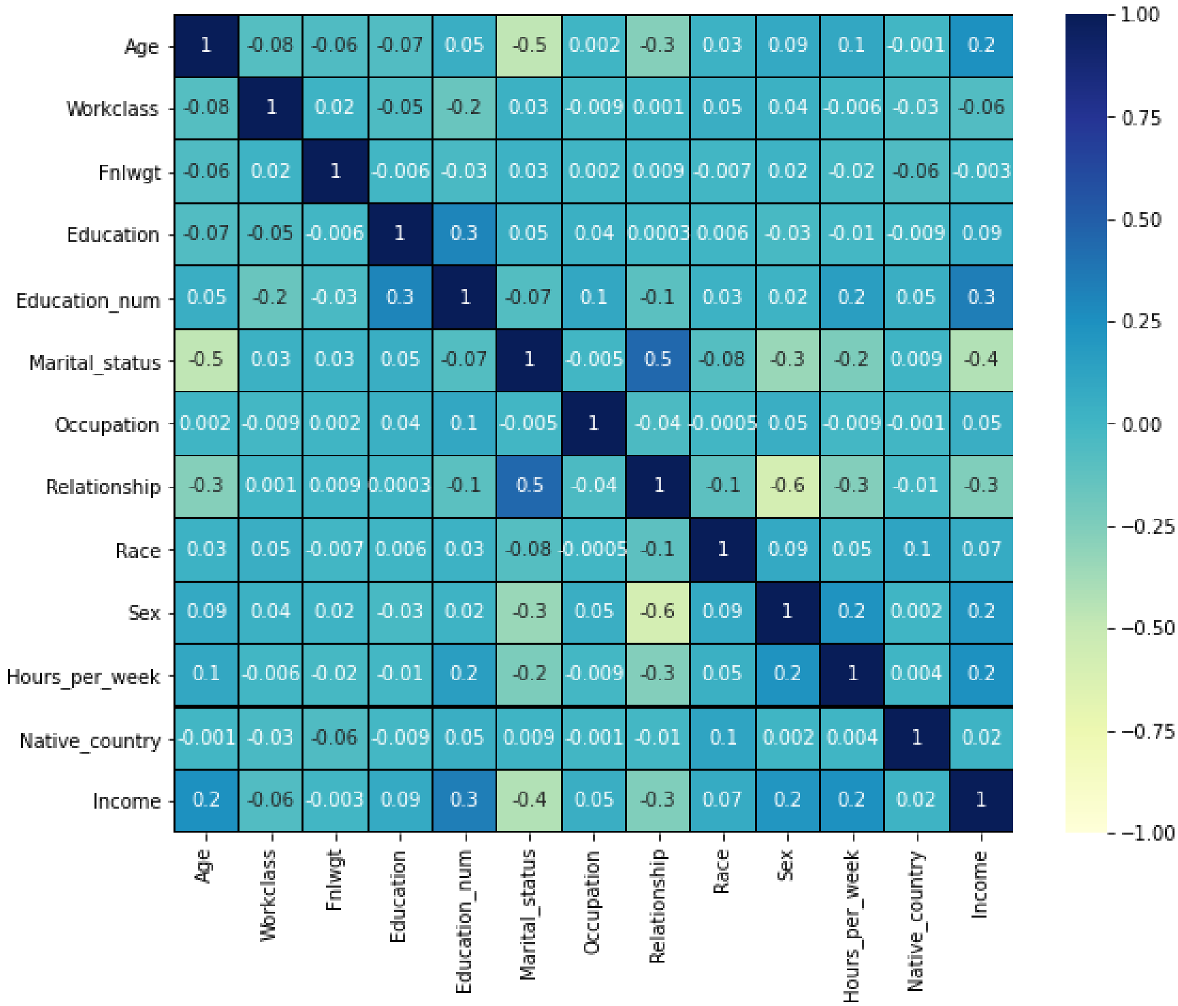
In [73]:
```
# checking the relation between features and the target
cor = new_df.corr()
cor
```

Out[73]:

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occ
Age	1.000000	-0.083618	-0.062328	-0.068447	0.053361	-0.476050	0
Workclass	-0.083618	1.000000	0.021707	-0.049975	-0.161488	0.034265	-0
Fnlwgt	-0.062328	0.021707	1.000000	-0.006265	-0.031874	0.030462	0
Education	-0.068447	-0.049975	-0.006265	1.000000	0.310261	0.052972	0
Education_num	0.053361	-0.161488	-0.031874	0.310261	1.000000	-0.071406	0
Marital_status	-0.476050	0.034265	0.030462	0.052972	-0.071406	1.000000	-0
Occupation	0.001946	-0.008539	0.001691	0.044614	0.098277	-0.005145	1
Relationship	-0.268028	0.001008	0.009060	0.000265	-0.102497	0.451130	-0
Race	0.030679	0.051670	-0.006959	0.006002	0.030849	-0.081701	-0
Sex	0.091664	0.036158	0.023307	-0.028825	0.016662	-0.336209	0
Hours_per_week	0.097510	-0.006349	-0.015820	-0.012020	0.160483	-0.241789	-0
Native_country	-0.001039	-0.031665	-0.061390	-0.009356	0.054510	0.009096	-0
Income	0.248351	-0.062963	-0.002780	0.085741	0.337595	-0.425904	0

- This gives the correlation between the dependent and independent variables. We can visualize this by ploting heat map.

In [74]:
```
# visualizing the correlation matrix by plotting heat map
plt.figure(figsize=(10,8))
sns.heatmap(new_df.corr(),linewidths=.1, vmin=-1, vmax=1, fmt='.1g', annot = Tr
plt.yticks(rotation=0);
```



This heatmap shows the correlation matrix by visualizing the data. We can observe the relation between one feature to other.

- This heat map contains both positive and negative correlaion
- There is no much correlation between the target and the label.
- The columns Education_num, Age, Sex and Hours_per_week have positive correlation with the target.
- The columns Marital status and Relationship have less correlation with the label.
- The columns Relationship and Sex are highly correlated with each other also the columns Fnlwgt has very less relation with the label so we can drop these columns if necessary.
- There is no multicolinearity issue exists in the dat aso no need to worry much.

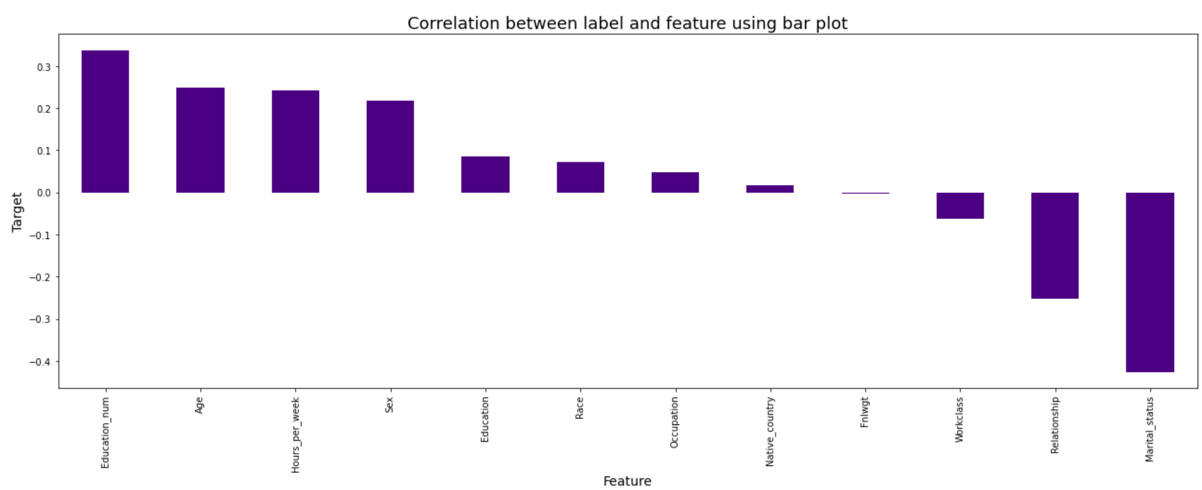
In [75]:
```
cor['Income'].sort_values(ascending=False)
```

Out[75]:
```
Income            1.000000
Education_num     0.337595
Age               0.248351
Hours_per_week    0.242383
Sex               0.216665
Education         0.085741
Race              0.072451
Occupation        0.048110
Native_country    0.017698
Fnlwgt           -0.002780
Workclass        -0.062963
Relationship     -0.251506
Marital_status   -0.425904
Name: Income, dtype: float64
```

- Here we can easily find the positive and negative correlation between features and the label.

Visualizing the correlation between label and features using bar plot

In [76]:
```
plt.figure(figsize=(22,7))
new_df.corr()['Income'].sort_values(ascending=False).drop(['Income']).plot(kinc
plt.xlabel('Feature',fontsize=14)
plt.ylabel('Target',fontsize=14)
plt.title('Correlation between label and feature using bar plot',fontsize=18)
plt.show()
```



- The columns Fnlwgt has very less correlation with the label so we can drop it if necessary.

Separating the features and label variables into

.

In [77]:
```
x = new_df.drop('Income', axis=1)
y = new_df['Income']
```

In [78]:
```
x.shape
```

Out[78]: (31461, 12)

In [79]:
```
y.shape
```

Out[79]: (31461,)

- Here we can see the dimension of y

Feature Scaling using Standard Scalarization

In [80]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
x
```

Out[80]:

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	R
0	0.875057	0.392103	-1.102052	1.245592	1.154324	-0.990371	-0.792500	
1	-0.025350	0.392103	0.464976	-1.277429	-0.459657	0.147158	-0.334828	
2	1.100158	0.392103	0.630260	0.236383	-1.266647	-0.990371	-0.334828	
3	-0.775689	0.392103	1.399791	1.245592	1.154324	-0.990371	0.809353	
4	-0.100384	0.392103	1.023711	-0.268221	1.557819	-0.990371	-0.792500	
...	...	...	...	...	...	...	...	
31456	-0.850723	0.392103	0.814728	0.740988	0.750829	-0.990371	1.495862	
31457	0.124718	0.392103	-0.143169	-1.277429	-0.459657	-0.990371	-0.105992	
31458	1.475327	0.392103	-0.170768	-1.277429	-0.459657	0.147158	-1.479009	
31459	-1.225892	0.392103	0.335900	-1.277429	-0.459657	1.284688	-1.479009	
31460	1.025124	0.392103	1.048402	-1.277429	-0.459657	-0.990371	-0.792500	

31461 rows × 12 columns

- So here we have scaled the data using standard scalarization method to overcome with the issue of data biasness.

Oversampling

In [81]:
```
from imblearn.over_sampling import SMOTE
SM = SMOTE()
x, y = SM.fit_resample(x,y)
```

In [82]:
```
y.value_counts()
```

Out[82]:
```
1    23853
0    23853
Name: Income, dtype: int64
```

In [83]:
```
# dataframe after preprocessing and data cleaning
new_df.head()
```

Out[83]:

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationshi
0	50	1	43.675121	5	13	0	3	
1	38	1	59.967204	0	9	1	5	
2	53	1	61.685627	3	7	0	5	
3	28	1	69.686283	5	13	0	10	
4	37	1	65.776255	2	14	0	3	

- We have done with the preprocessing and data cleaning. Now let's move to build the model.

Modeling

Finding best random state

In [84]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
maxAccu=0
maxRS=0
for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30, random_st
    DTC = DecisionTreeClassifier()
    DTC.fit(x_train,y_train)
    pred = DTC.predict(x_test)
    acc =accuracy_score(y_test,pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print ("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

```
Best accuracy is  0.839854667411962  on Random_state  125
```

- The best accuracy is 83.98% on the Random State 125

Creating train_test_split

In [85]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=m
```

Classification Algorithm

In [86]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,
```

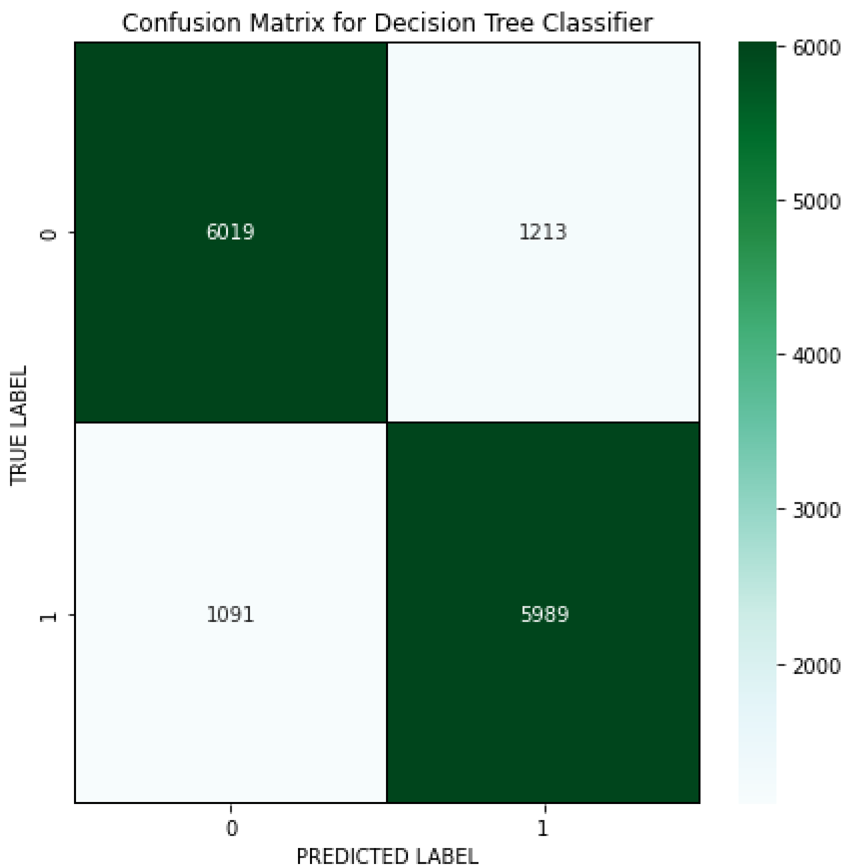
Decision Tree Classifier

In [87]:
```
DTC = DecisionTreeClassifier()
DTC.fit(x_train,y_train)

#Prediction
predDTC = DTC.predict(x_test)

print(accuracy_score(y_test,predDTC))
print(confusion_matrix(y_test,predDTC))
print(classification_report(y_test,predDTC))
```

```
0.8390162101732812
[[6019 1213]
 [1091 5989]]
              precision    recall  f1-score   support

           0       0.85      0.83      0.84      7232
           1       0.83      0.85      0.84      7080

    accuracy                           0.84     14312
   macro avg       0.84      0.84      0.84     14312
weighted avg       0.84      0.84      0.84     14312
```

- The accuracy using Decision Tree Classification is 83%

In [88]:
```
# let's plot confusion matrix for DTC
cm = confusion_matrix(y_test,predDTC)

x_axis_labels = ["0","1"]
y_axis_labels = ["0","1"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True, linewidths=.2, linecolor='black', fmt = '.0f', ax

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title("Confusion Matrix for Decision Tree Classifier")
plt.show()
```



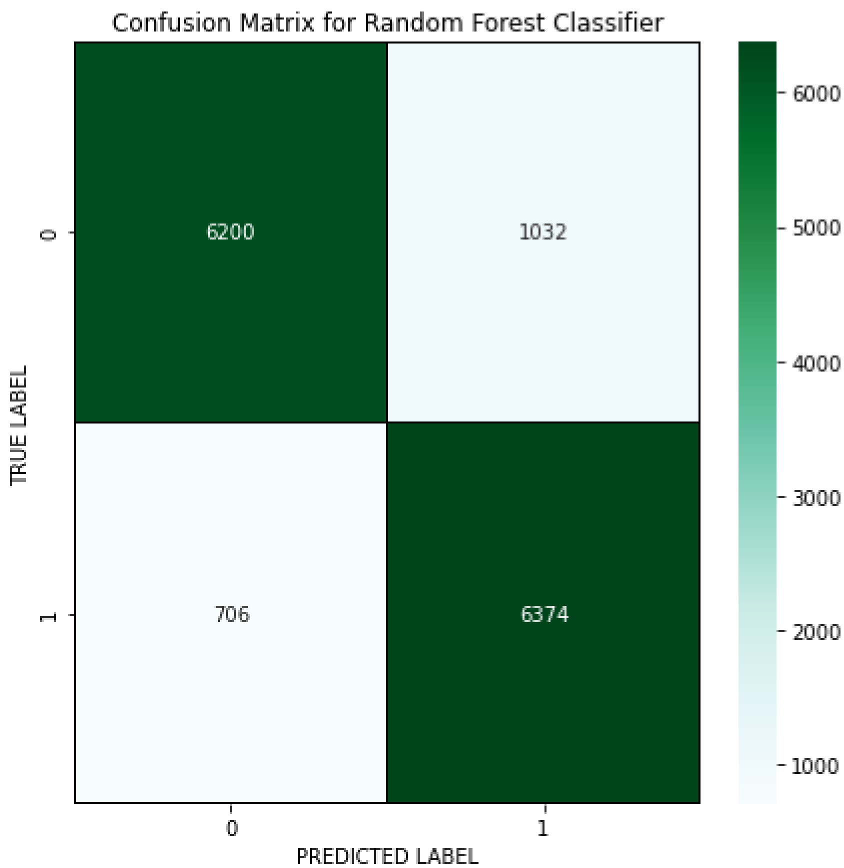
Random Forest Classifier

In [89]:
```
RFC = RandomForestClassifier()
RFC.fit(x_train,y_train)

#Prediction
predRFC = RFC.predict(x_test)

print(accuracy_score(y_test,predRFC))
print(confusion_matrix(y_test,predRFC))
print(classification_report(y_test,predRFC))
```

```
0.8785634432643935
[[6200 1032]
 [ 706 6374]]
              precision    recall  f1-score   support

           0       0.90      0.86      0.88      7232
           1       0.86      0.90      0.88      7080

    accuracy                           0.88     14312
   macro avg       0.88      0.88      0.88     14312
weighted avg       0.88      0.88      0.88     14312
```

- The accuracy using Random Forest Classifier is 87%

In [90]:
```
# let's plot confusion matrix for RFC
cm = confusion_matrix(y_test,predRFC)

x_axis_labels = ["0","1"]
y_axis_labels = ["0","1"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True, linewidths=.2, linecolor='black', fmt = '.0f', ax

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title("Confusion Matrix for Random Forest Classifier")
plt.show()
```



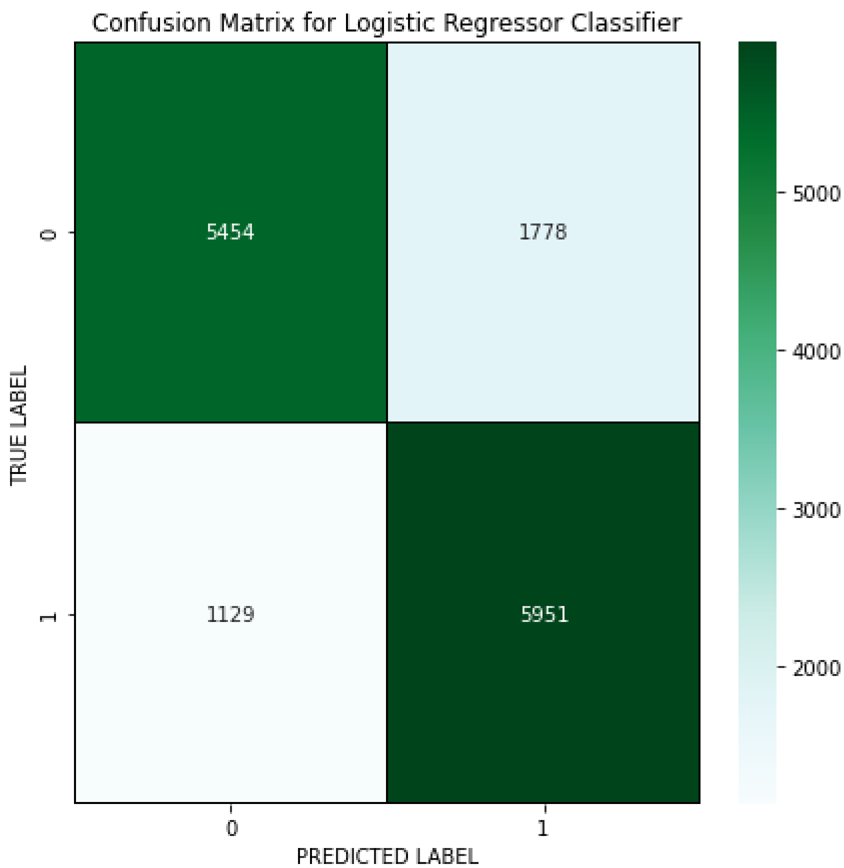
Logistic Regression

In [91]:
```
LR = LogisticRegression()
LR.fit(x_train,y_train)

#Prediction
predLR = LR.predict(x_test)

print(accuracy_score(y_test,predLR))
print(confusion_matrix(y_test,predLR))
print(classification_report(y_test,predLR))
```

```
0.7968837339295696
[[5454 1778]
 [1129 5951]]
              precision    recall  f1-score   support

           0       0.83      0.75      0.79      7232
           1       0.77      0.84      0.80      7080

    accuracy                           0.80     14312
   macro avg       0.80      0.80      0.80     14312
weighted avg       0.80      0.80      0.80     14312
```

- The accuracy using Logistic Regression is 79%

In [92]:
```
# let's plot confusion matrix for Logistic Regression
cm = confusion_matrix(y_test,predLR)

x_axis_labels = ["0","1"]
y_axis_labels = ["0","1"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True, linewidths=.2, linecolor='black', fmt = '.0f', ax

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title("Confusion Matrix for Logistic Regressor Classifier")
plt.show()
```



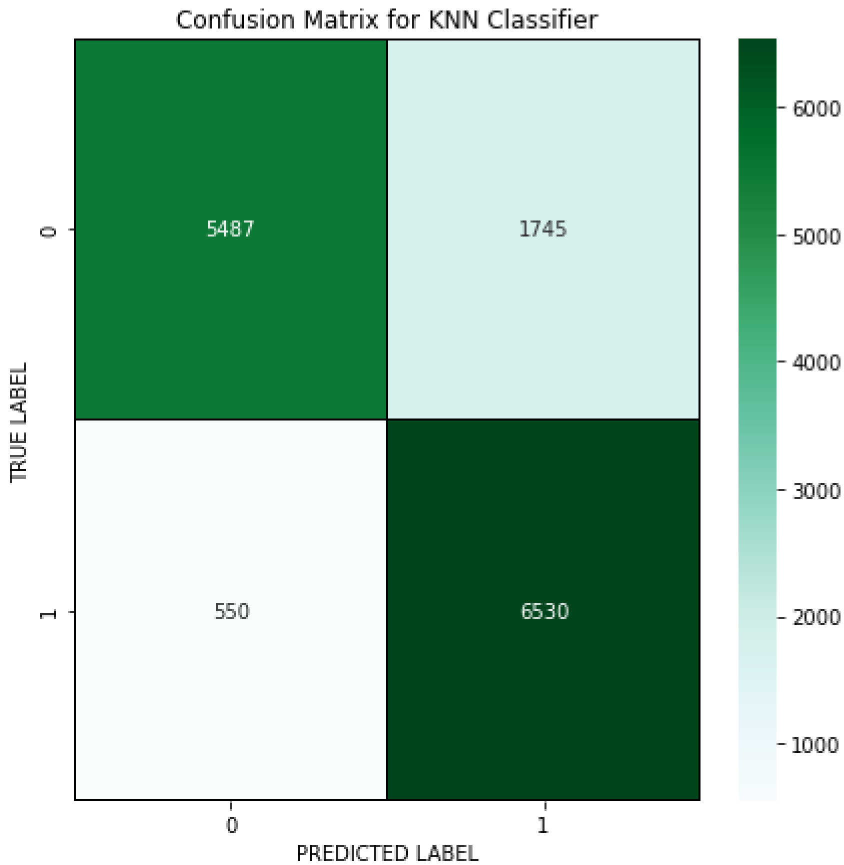
KNeighbores Classifier

In [93]:
```
knn = KNN()
knn.fit(x_train,y_train)

#Prediction
predknn = knn.predict(x_test)

print(accuracy_score(y_test,predknn))
print(confusion_matrix(y_test,predknn))
print(classification_report(y_test,predknn))
```

```
0.8396450531022918
[[5487 1745]
 [ 550 6530]]
              precision    recall  f1-score   support

           0       0.91      0.76      0.83      7232
           1       0.79      0.92      0.85      7080

    accuracy                           0.84     14312
   macro avg       0.85      0.84      0.84     14312
weighted avg       0.85      0.84      0.84     14312
```

- The accuracy using KNN is 83%

In [94]:
```
# let's plot confusion matrix for KNN Classifier
cm = confusion_matrix(y_test,predknn)

x_axis_labels = ["0","1"]
y_axis_labels = ["0","1"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True, linewidths=.2, linecolor='black', fmt = '.0f', ax

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title("Confusion Matrix for KNN Classifier")
plt.show()
```



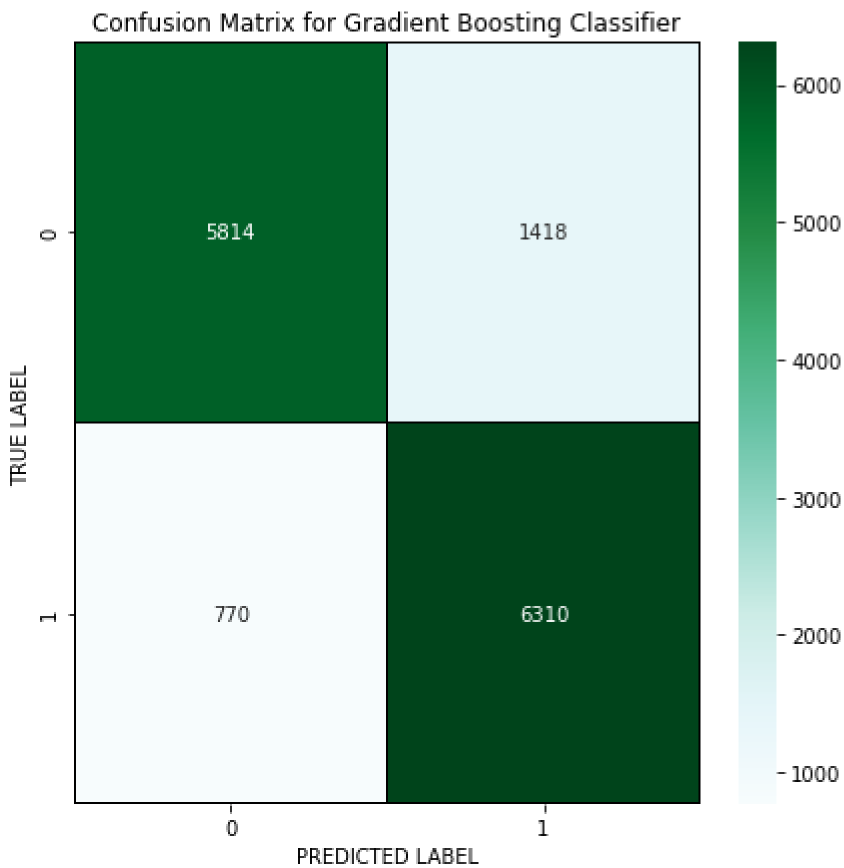
Gradient Boosting Classifier

In [95]:
```python
GB = GradientBoostingClassifier()
GB.fit(x_train,y_train)

#Prediction
predGB = GB.predict(x_test)

print(accuracy_score(y_test,predGB))
print(confusion_matrix(y_test,predGB))
print(classification_report(y_test,predGB))
```

```
0.8471212968138625
[[5814 1418]
 [ 770 6310]]
              precision    recall  f1-score   support

           0       0.88      0.80      0.84      7232
           1       0.82      0.89      0.85      7080

    accuracy                           0.85     14312
   macro avg       0.85      0.85      0.85     14312
weighted avg       0.85      0.85      0.85     14312
```

- The accuracy usin Gradient Boosting Classifier is 84%

In [96]:
```
# let's plot confusion matrix for Gradient Boosting Classifier
cm = confusion_matrix(y_test,predGB)

x_axis_labels = ["0","1"]
y_axis_labels = ["0","1"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True, linewidths=.2, linecolor='black', fmt = '.0f', ax

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title("Confusion Matrix for Gradient Boosting Classifier")
plt.show()
```



Checking the Cross Validation Score

In [97]:
```
from sklearn.model_selection import cross_val_score
```

In [98]:
```
# cv score for Decision Tree Classifier
print(cross_val_score(DTC,x,y,cv=5).mean())
```

```
0.8299604410015607
```

In [99]:
```
# cv score for Random Forest Classifier
print(cross_val_score(RFC ,x,y,cv=5).mean())
```

```
0.8815887311874085
```

In [100]:
```
# cv score for Lofistic Regression Classifier
print(cross_val_score(LR ,x,y,cv=5).mean())
```

```
0.7948267525094568
```

In [101]:
```
# cv score for KNN Classifier
print(cross_val_score( knn,x,y,cv=5).mean())
```

```
0.8483005193023365
```

In [102]:
```
# cv score for Gradient Boosting Classifier
print(cross_val_score(GB ,x,y,cv=5).mean())
```

```
0.8460160784757313
```

Above are the Cross Validation Score for the all models used

- From the difference between the accuracy score and the CV score we can conclude that Decision Tree Classifier as our best model.

Hyper Parameter Tuning

In [103]:
```
from sklearn.model_selection import GridSearchCV
```

In [104]:
```
# Decision Tree Classifier

parameters = {'criterion':['gini','entropy'],
             'max_features':['auto','sqrt','log2'],
             'max_depth':[10,20,30,40,50],
             'splitter':['best','random']}
```

In [105]:
```
GCV=GridSearchCV(DecisionTreeClassifier(),parameters,cv=5)
```

In [106]:
```
GCV.fit(x_train,y_train)
```

Out[106]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [10, 20, 30, 40, 50],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'splitter': ['best', 'random']})
```

```
In [107]: GCV.best_params_
```

```
Out[107]: {'criterion': 'entropy',
 'max_depth': 40,
 'max_features': 'auto',
 'splitter': 'best'}
```

```
In [108]: census = DecisionTreeClassifier(criterion='entropy', max_depth=20, max_features
census.fit(x_train,y_train)
pred = census.predict(x_test)
acc=accuracy_score(y_test,pred)
print(acc*100)
```

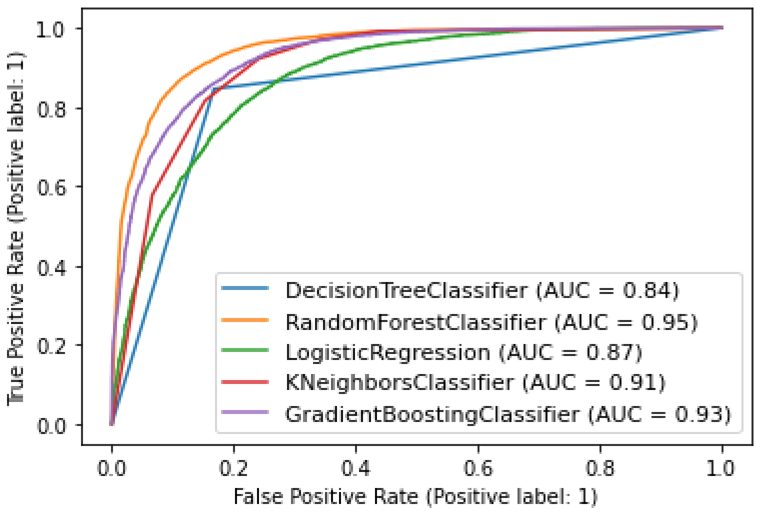
```
82.81162660704304
```

- So here we can see the accuracy of the best model is increased after tuning.

Plotting ROC and compare AUC for all the models used

In [109]:
```python
# plotting for all the models used here
from sklearn import datasets
from sklearn import metrics
from sklearn import model_selection
from sklearn.metrics import plot_roc_curve

disp = plot_roc_curve(DTC,x_test,y_test)
plot_roc_curve(RFC, x_test, y_test, ax=disp.ax_)
plot_roc_curve(LR, x_test, y_test, ax=disp.ax_)
plot_roc_curve(knn, x_test, y_test, ax=disp.ax_)
plot_roc_curve(GB, x_test, y_test, ax=disp.ax_)

plt.legend(prop={'size':11}, loc='lower right')
plt.show
```

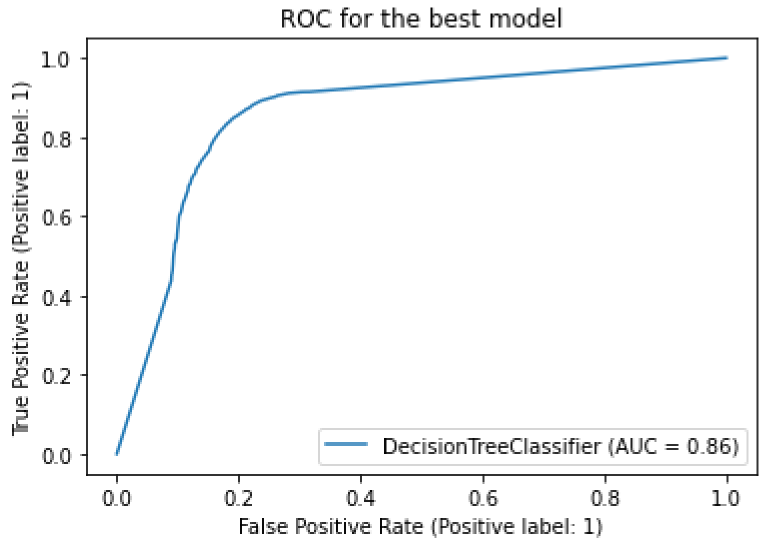
Out[109]: <function matplotlib.pyplot.show(close=None, block=None)>



- This is the AUC-ROC curve for the models that we have used and is plotted False positive rate against True Positive Rate.

Plotting ROC and Compare AUC for the best model

In [110]:
```
# let's check the Auc for the best model after hyper parameter tuning
plot_roc_curve(census, x_test, y_test)
plt.title("ROC for the best model")
plt.show()
```



- The AUC for the best model is 0.86

Saving The Model

In [111]:
```
import joblib
joblib.dump(census,"Census Income Prediction.pkl")
```

Out[111]:
```
['Census Income Prediction.pkl']
```

Predicting the saved model

In [112]:
```
# let's load the saved model and get the prediction

# loading the saved model
model=joblib.load("Census Income Prediction.pkl")

# prediction
prediction= model.predict(x_test)
prediction
```

Out[112]:
```
array([1, 0, 1, ..., 0, 1, 0])
```

In [113]:
```
pd.DataFrame([model.predict(x_test)[:],y_test[:]],index=['Predicted','Original'
```

Out[113]:

	Predicted	Original
0	1	1
1	0	0
2	1	1
3	0	0
4	1	0
...	...	...
14307	1	1
14308	0	0
14309	0	0
14310	1	1
14311	0	0

14312 rows × 2 columns

- So here we can see the predicted and actual values are almost same.

In [ ]:

In [ ]:

In [ ]:

In [ ]: