Adult Census Income

Author: Anthony Torres

Task: Predict whether income exceeds $50K/yr based on census data

Project task and source files were retrieved from the following Kaggle submission - https://www.kaggle.com/uciml/adult-census-income (https://www.kaggle.com/uciml/adult-census-income)

Dataset is based on this - https://archive.ics.uci.edu/ml/datasets/census+income (https://archive.ics.uci.edu/ml/datasets/census+income)

Data Set Information:

Extraction was done by Barry Becker from the 1994 Census database.

Listing of attributes:

- The last column >50K, <=50K is the target variable indicating whether the people earn less than or larger than 50K per year
- age: continuous.
- workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- fnlwgt: continuous.
- education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- education-num: continuous.
- marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- sex: Female, Male.
- capital-gain: continuous.
- capital-loss: continuous.
- hours-per-week: continuous.
- native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [3]:
```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

In [4]:
```
df = pd.read_csv("adult.csv")
df.head()
```

Out[4]:

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

Here's a brief overview of the steps i'm going to take:

1. EDA (chekcing missing values, removing outliers)
2. Perform basic exploration of relationship, with plots and graphs
3. Separate data set into training and testing
4. Setup dummy variables to take care categorical variables
5. Normalize numerical features if needed
6. Try at least two models and check their model performance
7. Perform cross-validations

First change the target variable income to 0 and 1

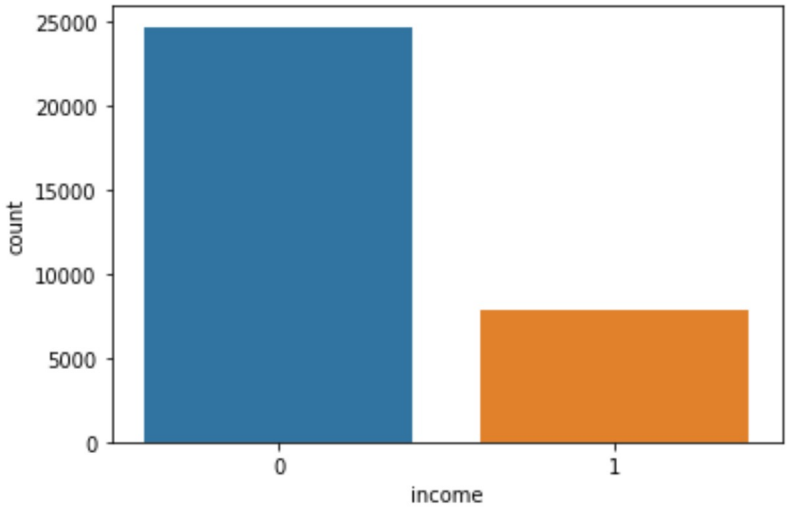
In [5]:
```
df['income'] = df['income'].apply(lambda x: 0 if x == '<=50K' else 1)
df.head()
```

Out[5]:

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

In [6]:
```
sns.countplot(df['income'])
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22393cb0d88>
```



Exploratory Data Analysis

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

Out[7]:

	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

In [8]:
```
df.shape
```

Out[8]:
```
(32561, 15)
```

In [9]:
```
#Checking for null/NAN values
df.isnull().any()
```

Out[9]:
```
age               False
workclass         False
fnlwgt            False
education         False
education.num     False
marital.status    False
occupation        False
relationship      False
race              False
sex               False
capital.gain      False
capital.loss      False
hours.per.week    False
native.country    False
income            False
dtype: bool
```

In [10]:
```
#Correlation Check
df.corr()
```

Out[10]:

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week
age	1.000000	-0.076646	0.036527	0.077674	0.057775	0.068756
fnlwgt	-0.076646	1.000000	-0.043195	0.000432	-0.010252	-0.018768
education.num	0.036527	-0.043195	1.000000	0.122630	0.079923	0.148123
capital.gain	0.077674	0.000432	0.122630	1.000000	-0.031615	0.078409
capital.loss	0.057775	-0.010252	0.079923	-0.031615	1.000000	0.054256
hours.per.week	0.068756	-0.018768	0.148123	0.078409	0.054256	1.000000
income	0.234037	-0.009463	0.335154	0.223329	0.150526	0.229689

In [11]:
```
# 'fnlwgt' has no relation to the other columns.
# Will drop the column
df = df.drop(['fnlwgt'], axis=1)
```

In [12]:
```
df.head(20)
```

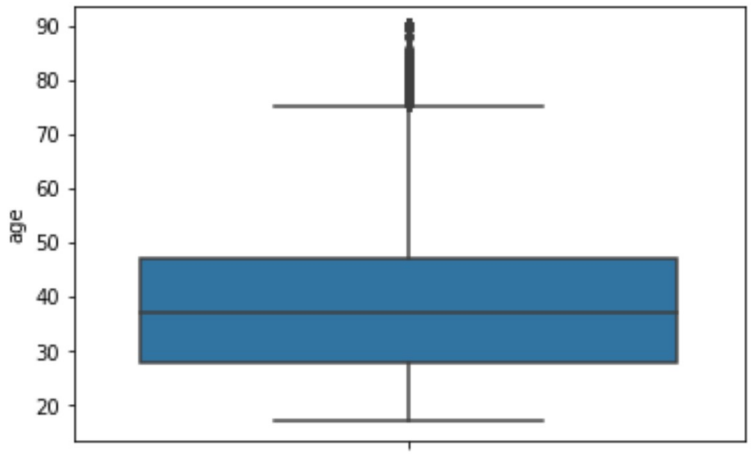
Out[12]:

	age	workclass	education	education.num	marital.status	occupation	relationship	race
0	90	?	HS-grad	9	Widowed	?	Not-in-family	White
1	82	Private	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White
2	66	?	Some-college	10	Widowed	?	Unmarried	Black
3	54	Private	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White
4	41	Private	Some-college	10	Separated	Prof-specialty	Own-child	White
5	34	Private	HS-grad	9	Divorced	Other-service	Unmarried	White
6	38	Private	10th	6	Separated	Adm-clerical	Unmarried	White
7	74	State-gov	Doctorate	16	Never-married	Prof-specialty	Other-relative	White
8	68	Federal-gov	HS-grad	9	Divorced	Prof-specialty	Not-in-family	White
9	41	Private	Some-college	10	Never-married	Craft-repair	Unmarried	White
10	45	Private	Doctorate	16	Divorced	Prof-specialty	Unmarried	Black
11	38	Self-emp-not-inc	Prof-school	15	Never-married	Prof-specialty	Not-in-family	White
12	52	Private	Bachelors	13	Widowed	Other-service	Not-in-family	White
13	32	Private	Masters	14	Separated	Exec-managerial	Not-in-family	White
14	51	?	Doctorate	16	Never-married	?	Not-in-family	White
15	46	Private	Prof-school	15	Divorced	Prof-specialty	Not-in-family	White
16	45	Private	11th	7	Divorced	Transport-moving	Not-in-family	White
17	57	Private	Masters	14	Divorced	Exec-managerial	Not-in-family	White
18	22	Private	Assoc-acdm	12	Never-married	Handlers-cleaners	Not-in-family	Black
19	34	Private	Bachelors	13	Separated	Sales	Not-in-family	White

In [13]:
```
# I will remove all categorical values with a "?" value
df = df[ df['native.country'] != "?"]
df = df[ df['workclass'] != "?"]
df = df[ df['education'] != "?"]
df = df[ df['race'] != "?"]
df = df[ df['relationship'] != "?"]
df = df[ df['sex'] != "?"]
df = df[ df['occupation'] != "?"]
df = df[ df['marital.status'] != "?"]
```

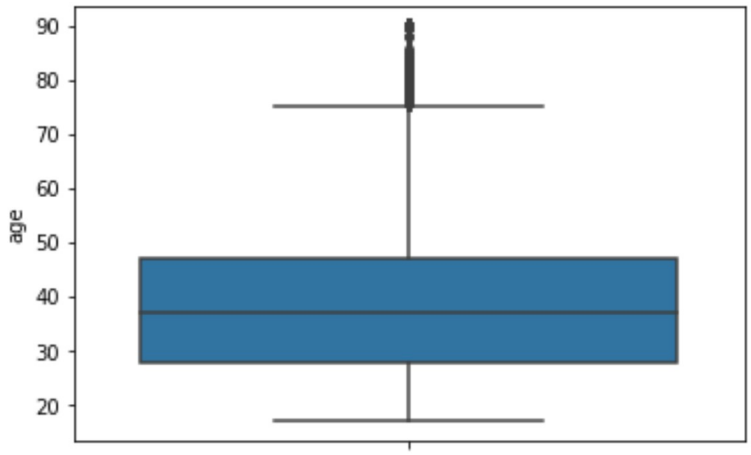
In [14]:
```
# Checking some outliers
sns.boxplot(y=df['age'])
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x22393babd08>



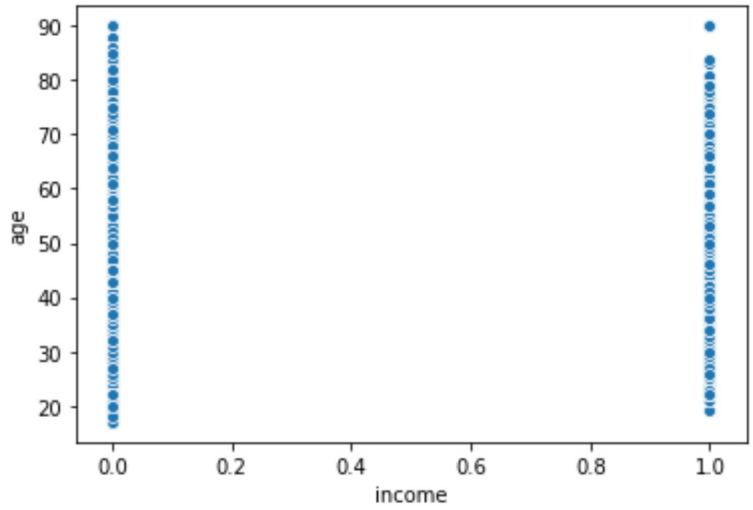
In [15]:
```
# Obviously shouldn't have ages 0 or above 150
# Will remove values >90 and <0
df = df[ df['age'] <= 90]
df = df[ df['age'] >0]
sns.boxplot(y=df['age'])
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x22393c25308>



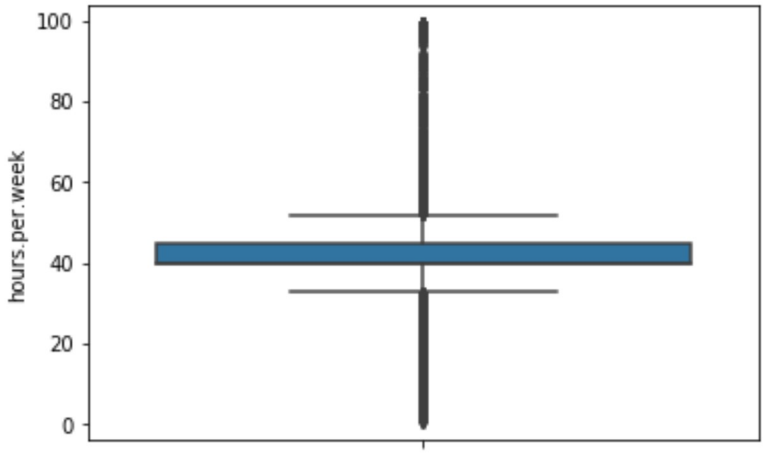
In [16]:
```
sns.scatterplot(x='income', y = 'age', data=df)
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x22393b48a48>



In [17]:
```
sns.boxplot(y=df['hours.per.week'])
```

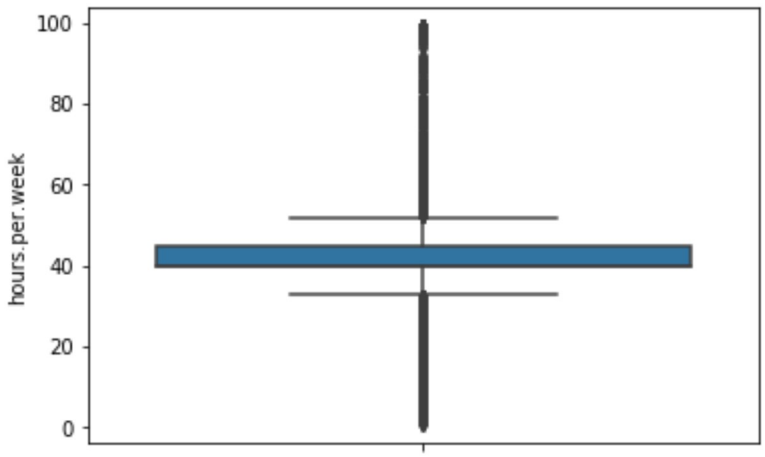
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x223945103c8>



In [18]:
```
#No way to have 200+ hrs per week.
df = df[ df['hours.per.week'] < 200]
```

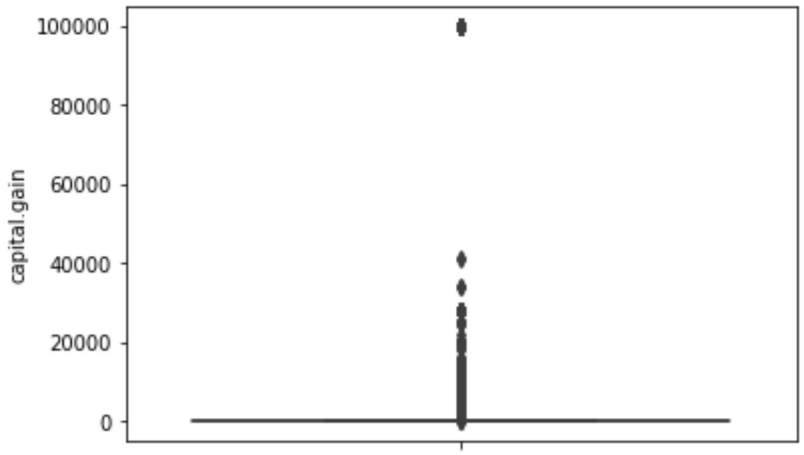
In [19]:
```
sns.boxplot(y=df['hours.per.week'])
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22394573948>
```



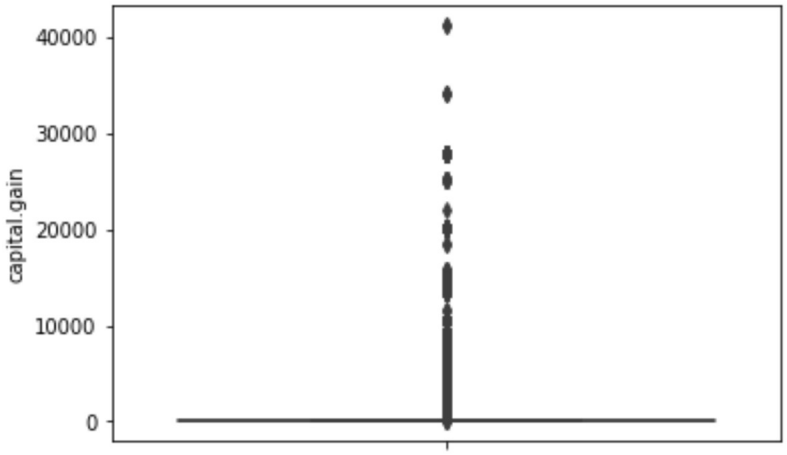
In [20]:
```
sns.boxplot(y=df['capital.gain'])
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x223945d0688>
```



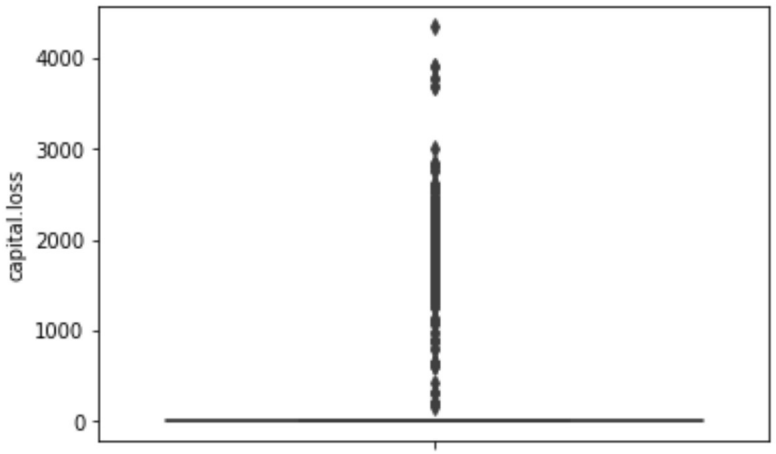
In [21]:
```
# Removing outliers from cap-gain
df = df[ df['capital.gain'] < 99999]
sns.boxplot(y=df['capital.gain'])
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x22394863f48>



In [22]:
```
sns.boxplot(y=df['capital.loss'])
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x22394a26f48>



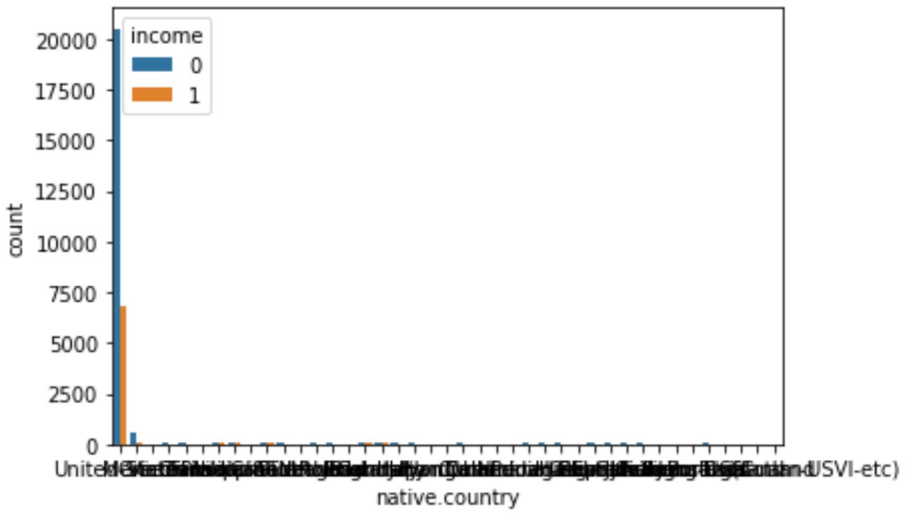
In [23]:
```
df.corr()
```

Out[23]:

	age	education.num	capital.gain	capital.loss	hours.per.week	income
age	1.000000	0.040166	0.119404	0.060958	0.100737	0.239325
education.num	0.040166	1.000000	0.146012	0.081312	0.148043	0.329919
capital.gain	0.119404	0.146012	1.000000	-0.051368	0.084356	0.309907
capital.loss	0.060958	0.081312	-0.051368	1.000000	0.053484	0.153087
hours.per.week	0.100737	0.148043	0.084356	0.053484	1.000000	0.225425
income	0.239325	0.329919	0.309907	0.153087	0.225425	1.000000

In [24]:
```
# Wanted to see how much % of the entries are from US
sns.countplot(df['native.country'], hue = df['income'])
```

Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22394a7e848>
```



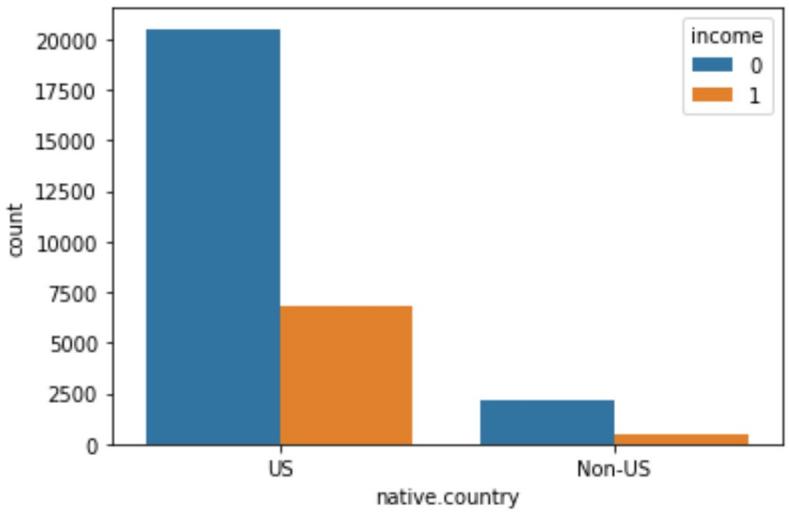
In [25]:
```
# Since majority is from US, will change into a binary (Non-US/US) valu
e
df['native.country'] = df['native.country'].apply(lambda x: 'US' if x =
= 'United-States' else 'Non-US')
df.head()
```

Out[25]:

	age	workclass	education	education.num	marital.status	occupation	relationship	race	
1	82	Private	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	F
3	54	Private	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	F
4	41	Private	Some-college	10	Separated	Prof-specialty	Own-child	White	F
5	34	Private	HS-grad	9	Divorced	Other-service	Unmarried	White	F
6	38	Private	10th	6	Separated	Adm-clerical	Unmarried	White	

In [26]:
```
sns.countplot(df['native.country'], hue = df['income'])
```

Out[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22394c66a48>
```



In [27]:
```
df.isnull().any()
```

Out[27]:
```
age               False
workclass         False
education         False
education.num     False
marital.status    False
occupation        False
relationship      False
race              False
sex               False
capital.gain      False
capital.loss      False
hours.per.week    False
native.country    False
income            False
dtype: bool
```

In [28]:
```
#Creating test/train split of (80/20)
#Creating train set
train = df.sample(int(df.shape[0] * 0.8))
print(train.shape)
train.head()
```

```
(24011, 14)
```

Out[28]:

	age	workclass	education	education.num	marital.status	occupation	relationship	ra
2631	46	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	Whi
531	23	Private	Assoc-acdm	12	Never-married	Adm-clerical	Own-child	Whi
15178	28	Private	HS-grad	9	Married-civ-spouse	Craft-repair	Husband	Whi
15608	30	Self-emp-not-inc	7th-8th	4	Married-civ-spouse	Farming-fishing	Husband	Whi
31304	46	Private	Some-college	10	Married-civ-spouse	Adm-clerical	Husband	Whi

In [29]:
```
testing_rows = [idx for idx in df.index if idx not in train.index]
len(testing_rows)
```

Out[29]: 6003

In [30]:
```
test_df = df.loc[testing_rows, :]
print(test_df.shape)
test_df.head()
```

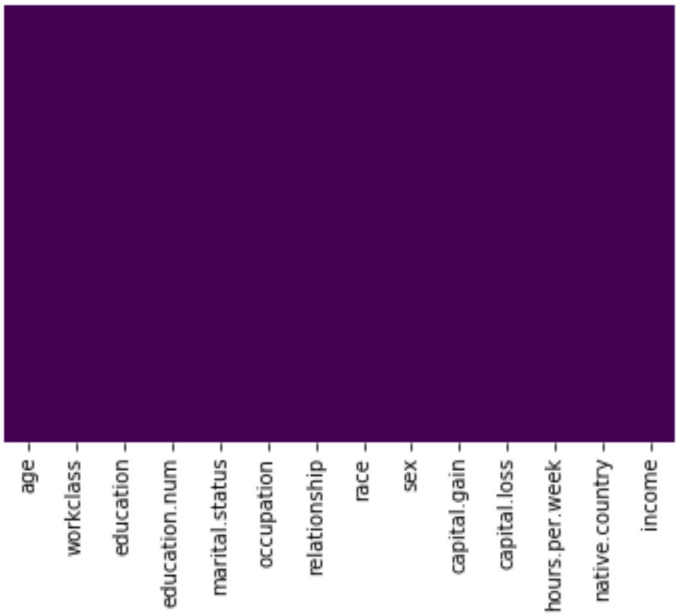
```
(6003, 14)
```

Out[30]:

	age	workclass	education	education.num	marital.status	occupation	relationship	race
7	74	State-gov	Doctorate	16	Never-married	Prof-specialty	Other-relative	White
10	45	Private	Doctorate	16	Divorced	Prof-specialty	Unmarried	Black
12	52	Private	Bachelors	13	Widowed	Other-service	Not-in-family	White
31	59	Self-emp-inc	10th	6	Widowed	Exec-managerial	Unmarried	White
32	52	Private	Prof-school	15	Divorced	Exec-managerial	Not-in-family	White

In [31]:
```
##EDA
sns.heatmap(train.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22394e7b488>
```



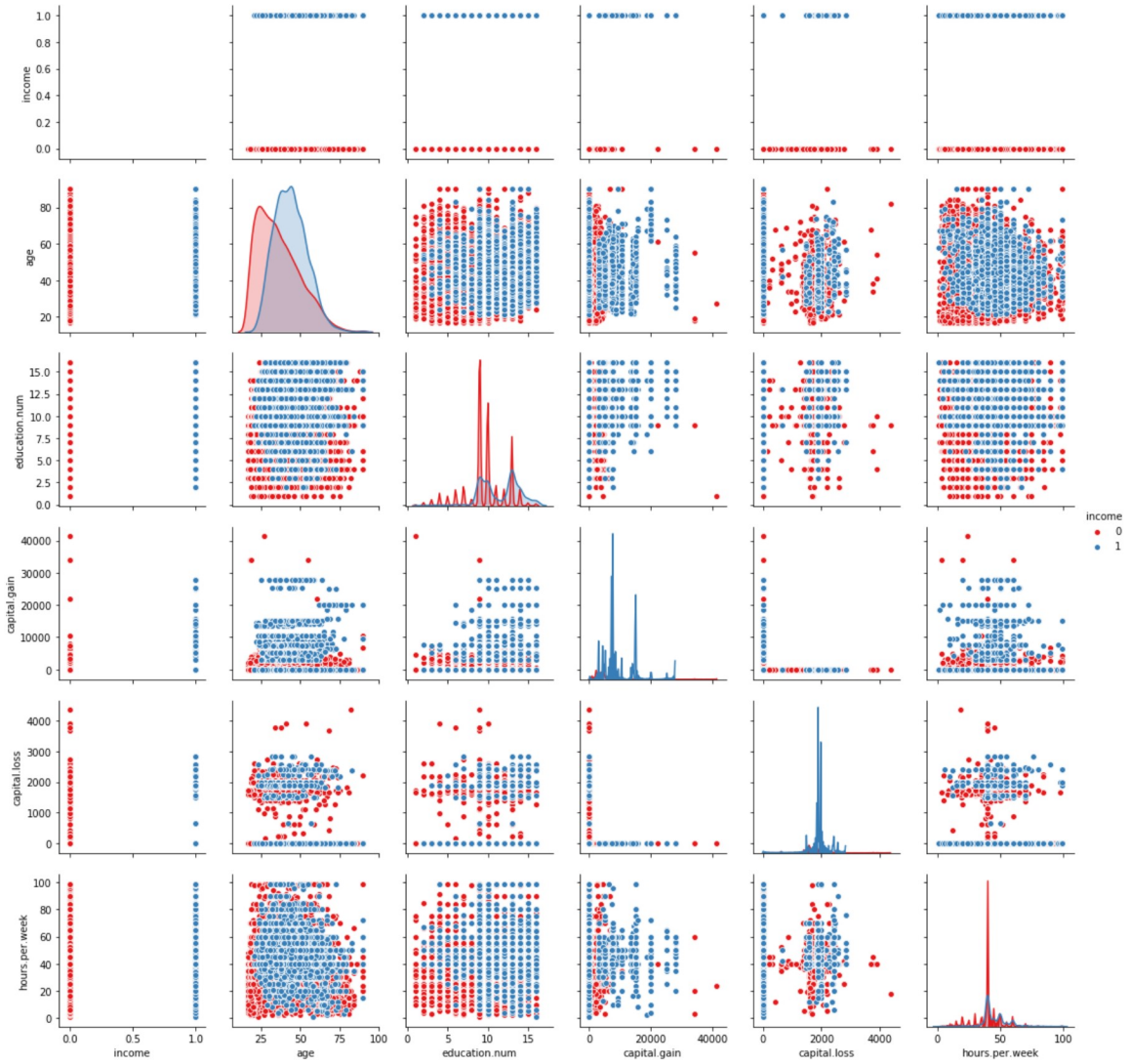
In [32]:
```
# Using just the numerical variables (will come back to revisit)
train_NUM = train[['income','age', 'education.num', 'capital.gain', 'capital.loss', 'hours.per.week']]
train_NUM.head()
```

Out[32]:

	income	age	education.num	capital.gain	capital.loss	hours.per.week
2631	1	46	13	7688	0	45
531	0	23	12	0	1974	20
15178	0	28	9	0	0	40
15608	0	30	4	0	0	40
31304	1	46	10	0	0	40

In [33]:
```
sns.pairplot(train_NUM, hue='income',palette='Set1')
```

Out[33]: <seaborn.axisgrid.PairGrid at 0x22394e47388>



In [34]:
```
#Normalizing cap-gain/cap-loss to a range of 0-10
train_NUM["capital.gain"]=((train_NUM["capital.gain"]-train_NUM["capital.gain"].min())/(train_NUM["capital.gain"].max()-train_NUM["capital.gain"].min()))*10
train_NUM["capital.loss"]=((train_NUM["capital.loss"]-train_NUM["capital.loss"].min())/(train_NUM["capital.loss"].max()-train_NUM["capital.loss"].min()))*10
train_NUM.describe()
```

Out[34]:

	income	age	education.num	capital.gain	capital.loss	hours.per.week
count	24011.000000	24011.000000	24011.000000	24011.000000	24011.000000	24011.000000
mean	0.245221	38.401816	10.107076	0.147910	0.204624	40.862896
std	0.430227	13.116734	2.549140	0.628497	0.931957	11.971817
min	0.000000	17.000000	1.000000	0.000000	0.000000	1.000000
25%	0.000000	28.000000	9.000000	0.000000	0.000000	40.000000
50%	0.000000	37.000000	10.000000	0.000000	0.000000	40.000000
75%	0.000000	47.000000	12.000000	0.000000	0.000000	45.000000
max	1.000000	90.000000	16.000000	10.000000	10.000000	99.000000

Creating a Logistic Regression model

In [35]:
```
X_train, X_test, y_train, y_test = train_test_split(train_NUM.drop('income',axis=1),
                                                    train_NUM['income'], test_size=0.20,
                                                    random_state=101)
model6 = LogisticRegression()
model6.fit(X_train,y_train)
predictions = model6.predict(X_test)
print(classification_report(y_test,predictions))
print(accuracy_score(y_test, predictions))
```

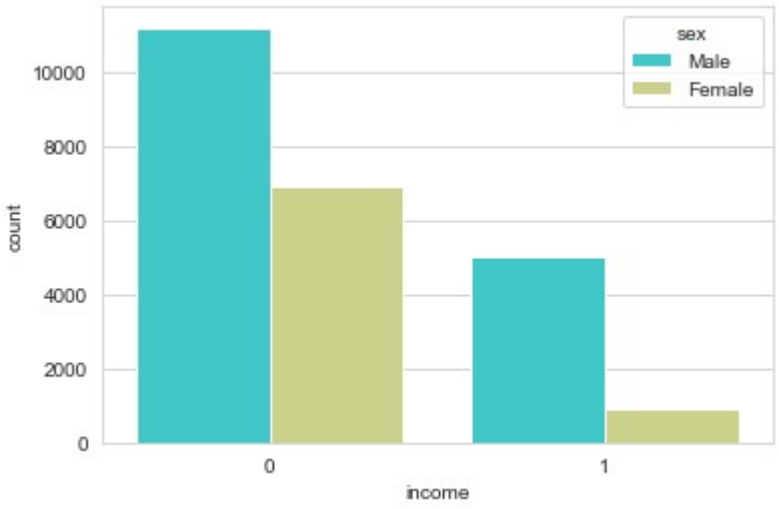
```
              precision    recall  f1-score   support

           0       0.83      0.95      0.89      3657
           1       0.70      0.38      0.50      1146

    accuracy                           0.81      4803
   macro avg       0.77      0.67      0.69      4803
weighted avg       0.80      0.81      0.79      4803

0.8138663335415366
```

Got accuracy of .81

Would like to check the relationships for the categorical columns.

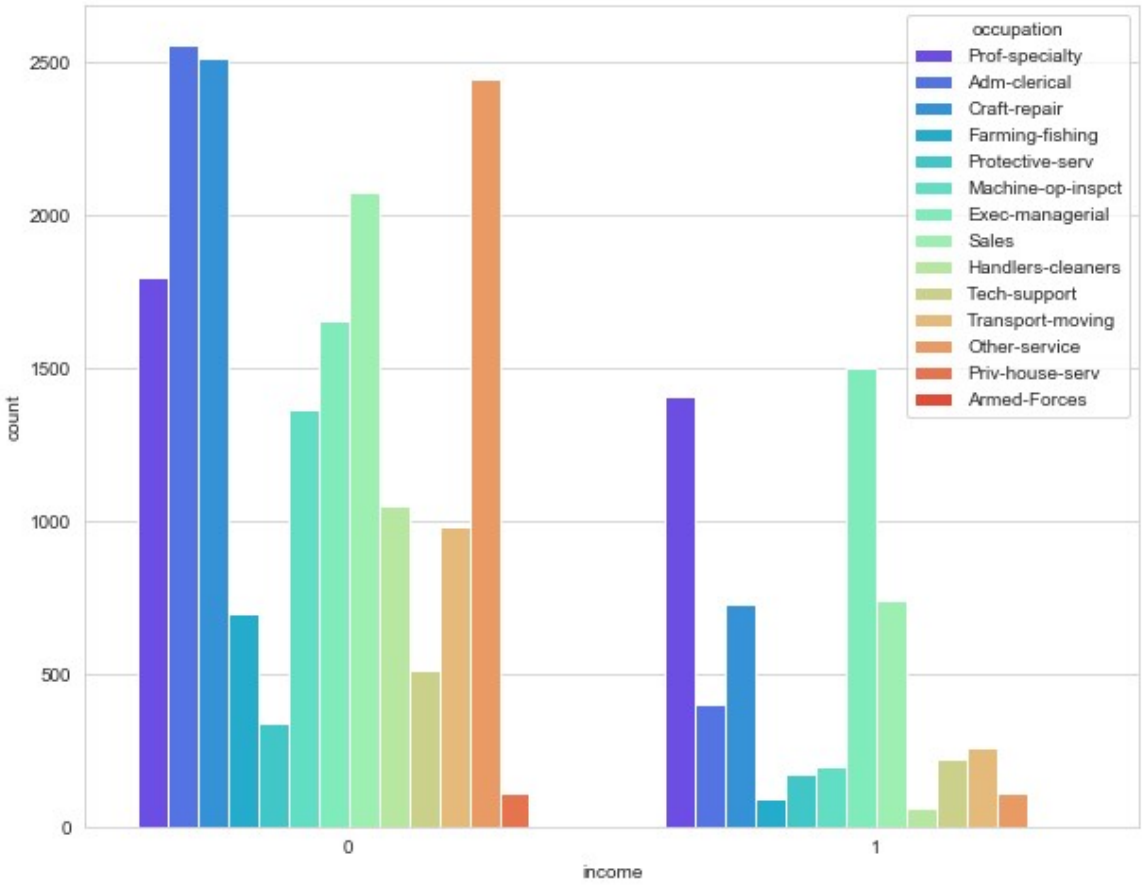
In [36]:
```
sns.set_style('whitegrid')
sns.countplot(x='income',hue='sex',data=train,palette='rainbow')
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22399b82a88>
```



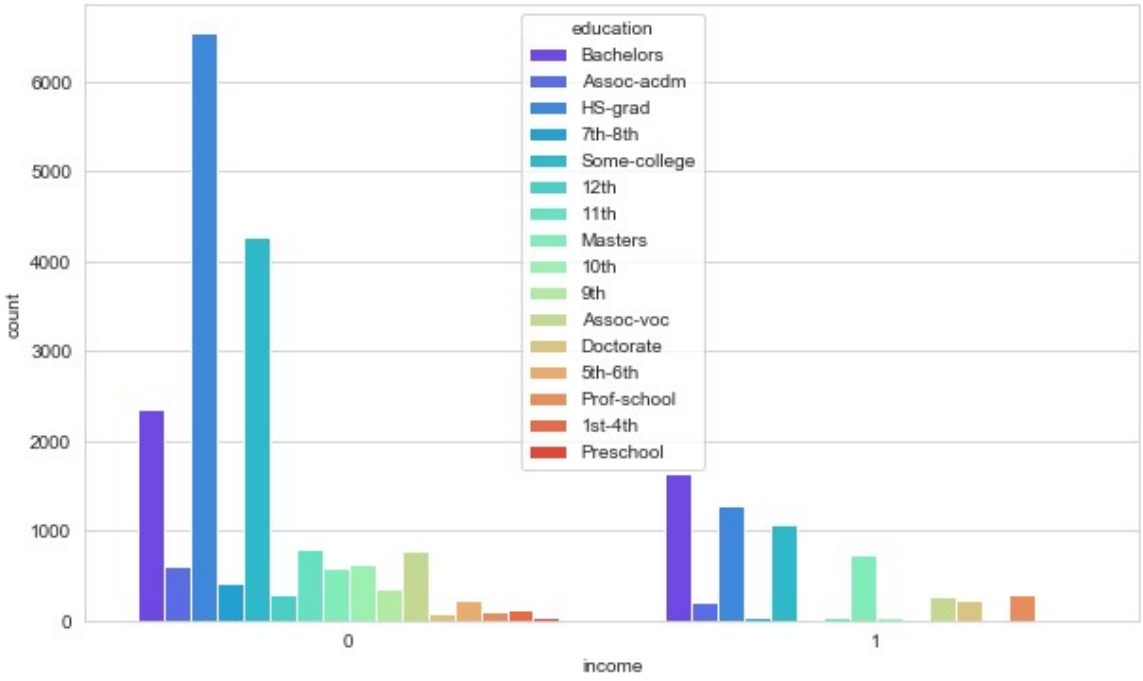
In [37]:
```
fig_dims = (10, 8)
fig, ax = plt.subplots(figsize=fig_dims)
sns.countplot(x='income',hue='occupation',data=train,palette='rainbow',
ax=ax)
```

Out[37]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2239b03d048>
```



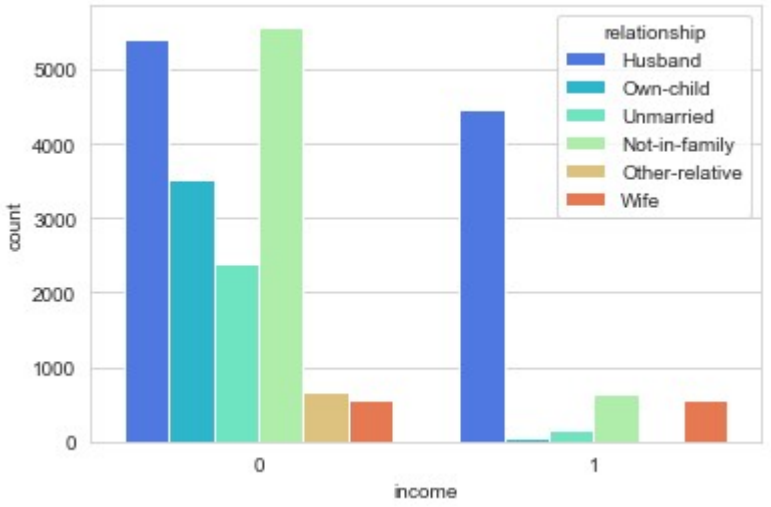
In [38]:
```
fig, ax = plt.subplots(figsize=(10,6))
sns.countplot(x='income',hue='education',data=train,palette='rainbow',
ax=ax)
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x2239b2cb748>



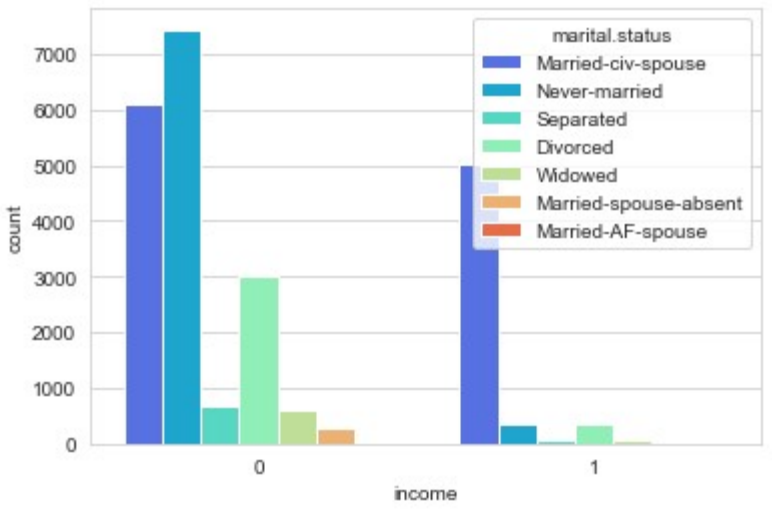
In [39]:
```
sns.countplot(x='income',hue='relationship',data=train,palette='rainbow
')
```

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x2239b1ab548>



In [40]:
```
sns.countplot(x='income',hue='marital.status',data=train,palette='rainbow')
```

Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2239b248208>
```



The ratio for marital-status is very high

So is the one for gender

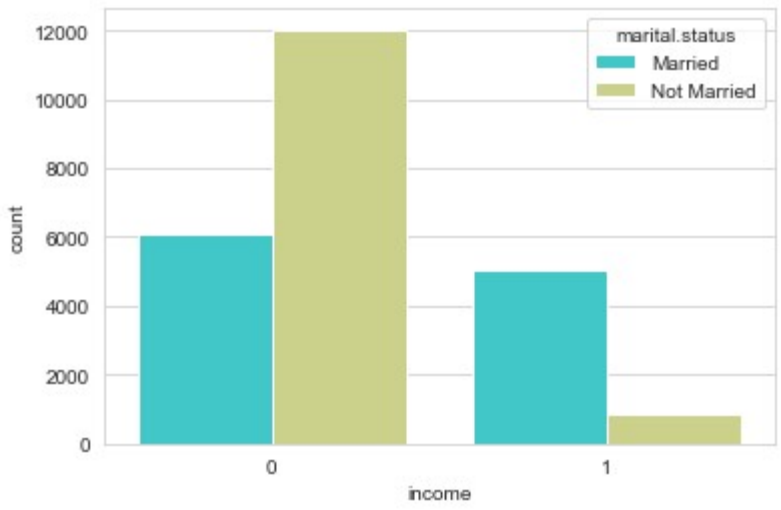
In [41]:
```
df_mar = train[['income', 'marital.status']]
df_mar['marital.status'] = df_mar['marital.status'].apply(lambda x: 'Married' if (x == 'Married-civ-spouse') else 'Not Married')
df_mar.head()
```

Out[41]:

	income	marital.status
2631	1	Married
531	0	Not Married
15178	0	Married
15608	0	Married
31304	1	Married

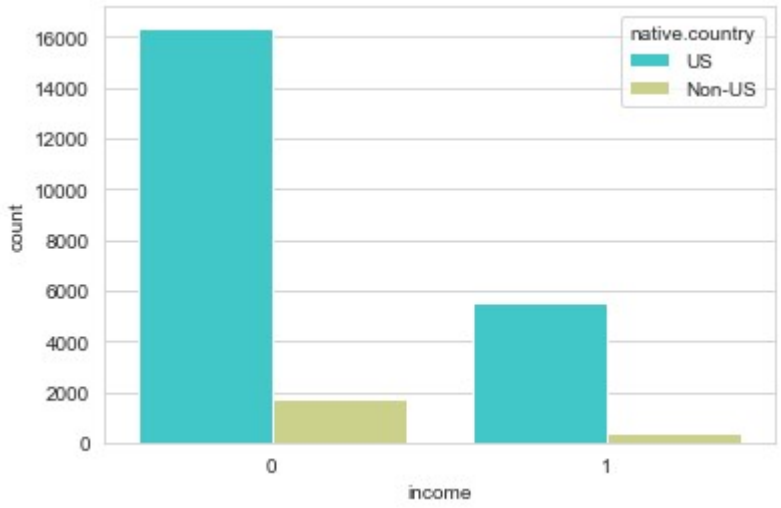
In [42]:
```
sns.countplot(x='income',hue='marital.status',data=df_mar,palette='rainbow')
```

Out[42]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2239b4968c8>
```



In [43]:
```
sns.countplot(x='income',hue='native.country',data=train,palette='rainbow')
```

Out[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2239b4f1788>
```



In [44]:
```
#Will make an attempt to include these in the model
sex = pd.get_dummies(train['sex'],drop_first=True)
```

In [45]:
```
married = pd.get_dummies(df_mar['marital.status'],drop_first=True)
married
```

Out[45]:

	Not Married
2631	0
531	1
15178	0
15608	0
31304	0
...	...
23385	1
14937	1
6979	1
4308	0
9494	1

24011 rows × 1 columns

In [46]:
```
train1 = pd.concat([train_NUM, sex, married], axis=1)
```

In [47]:
```
train1.describe()
```

Out[47]:

	income	age	education.num	capital.gain	capital.loss	hours.per.week
count	24011.000000	24011.000000	24011.000000	24011.000000	24011.000000	24011.000000
mean	0.245221	38.401816	10.107076	0.147910	0.204624	40.862896
std	0.430227	13.116734	2.549140	0.628497	0.931957	11.971817
min	0.000000	17.000000	1.000000	0.000000	0.000000	1.000000
25%	0.000000	28.000000	9.000000	0.000000	0.000000	40.000000
50%	0.000000	37.000000	10.000000	0.000000	0.000000	40.000000
75%	0.000000	47.000000	12.000000	0.000000	0.000000	45.000000
max	1.000000	90.000000	16.000000	10.000000	10.000000	99.000000

In [48]:
```
train1.isnull().any()
```

Out[48]:
```
income            False
age               False
education.num     False
capital.gain      False
capital.loss      False
hours.per.week    False
Male              False
Not Married       False
dtype: bool
```

Another Logistic Regression Model

In [49]:
```
X_train, X_test, y_train, y_test = train_test_split(train1.drop('income
',axis=1),
                                                    train1['income'], t
est_size=0.20,
                                                    random_state=101)
model1 = LogisticRegression()
model1.fit(X_train,y_train)
predictions = model1.predict(X_test)
print(classification_report(y_test,predictions))
print(accuracy_score(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.87      0.93      0.90      3657
           1       0.71      0.55      0.62      1146

    accuracy                           0.84      4803
   macro avg       0.79      0.74      0.76      4803
weighted avg       0.83      0.84      0.83      4803

0.8380179054757443
```

Got .84 accuracy

10 Fold Cross-Validation

In [50]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model1, X_train, y_train, cv=10)
print('Cross-Validation Accuracy Scores', scores)
```

```
Cross-Validation Accuracy Scores [0.8433923  0.84027055 0.8292556  0.
83133784 0.83550234 0.83029672
 0.83854167 0.83541667 0.84791667 0.83802083]
```

In [51]:
```
scores.mean()
```

Out[51]:
```
0.8369951188434219
```

.8369, close

Decision trees

In [52]:
```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=3)
tree.fit(X_train,y_train)
```

Out[52]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth
=3,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=
None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [53]:
```
Tpredictions = tree.predict(X_test)
print(confusion_matrix(y_test,Tpredictions))
print(classification_report(y_test,Tpredictions))
print(accuracy_score(y_test, Tpredictions))
```

```
[[3458  199]
 [ 589  557]]
              precision    recall  f1-score   support

           0       0.85      0.95      0.90      3657
           1       0.74      0.49      0.59      1146

    accuracy                           0.84      4803
   macro avg       0.80      0.72      0.74      4803
weighted avg       0.83      0.84      0.82      4803

0.8359358734124506
```

Around .84 Accuracy

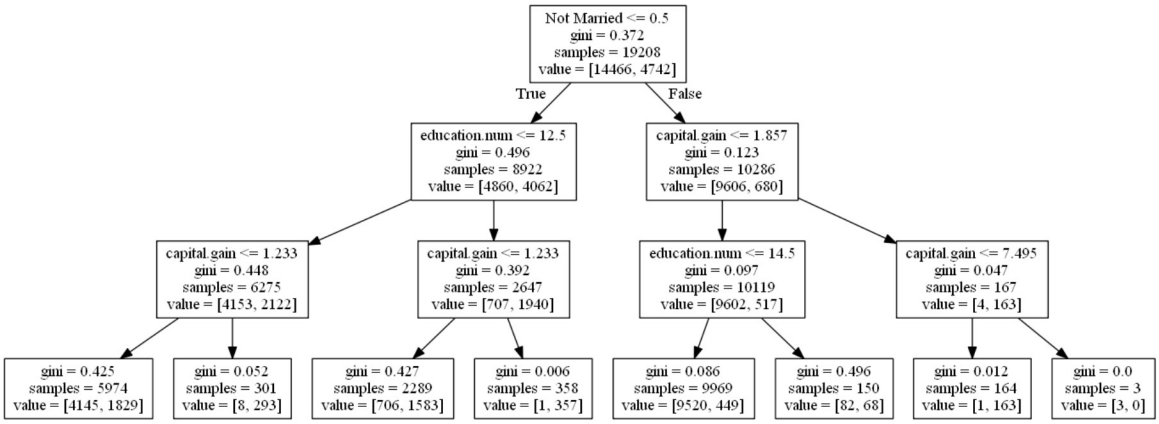
In [54]:
```
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import io
import pydot

features = list(train1.columns[1:])
features
```

```
C:\Users\23267958\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\six.py:31: DeprecationWarning: The module is deprecated in version 0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Please rely on the official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

Out[54]:
```
['age',
 'education.num',
 'capital.gain',
 'capital.loss',
 'hours.per.week',
 'Male',
 'Not Married']
```

In [55]:
```
dot_data = StringIO()

export_graphviz(tree, out_file=dot_data, feature_names=features)
(graph, ) = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[55]:

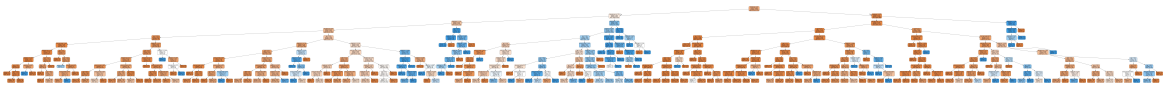


Changing max_depth to 10

In [56]:
```
Tmodel = DecisionTreeClassifier(max_depth=10, criterion='entropy')
Tmodel.fit(X_train,y_train)
predictions = Tmodel.predict(X_test)
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
print(accuracy_score(y_test, predictions))

dot_data = StringIO()
export_graphviz(Tmodel, out_file=dot_data,feature_names=features,filled
=True,rounded=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
```

```
[[3502  155]
 [ 572  574]]
              precision    recall  f1-score   support

           0       0.86      0.96      0.91      3657
           1       0.79      0.50      0.61      1146

    accuracy                           0.85      4803
   macro avg       0.82      0.73      0.76      4803
weighted avg       0.84      0.85      0.84      4803

0.8486362689985426
```

Out[56]:



Around .85 Accuracy

Highest Score: 0.8486362689985426, Decision Tree