In [1]:
```
#Packages
import numpy as np
import pandas as pd

import numpy.random as nr
from pandas import Series, DataFrame
import matplotlib as mpl
import matplotlib.pyplot as plt
from pylab import plot, show

from matplotlib import rcParams
import seaborn as sns


# Ignore warnings
import warnings
warnings.filterwarnings('ignore');
```

In [2]:
```
import pandas as pd
columns = ['Age', 'Work Class', 'Final Weight', 'Education', 'Education Numbe
r', 'Marital Status', 'Occupation',
          'Relationship', 'Race', 'Sex', 'Capital Gain', 'Capital Loss', 'Hour
s per Week', 'Country', 'Income']
census_data = pd.read_csv('adult.data', names = columns)
```

In [3]:
```
pd.set_option("max_rows", None)
census_data.head(5)
```

Out[3]:

	Age	Work Class	Final Weight	Education	Education Number	Marital Status	Occupation	Relationship	Race	Se
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Ma
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Ma
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Ma
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Ma
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Fema

Target Variable - Income (y)

In order to build a predictive model whether an individual's income exceeds $50K/year based on census data (classification), the target column is "income" where the labels are classified as <=50K and >50K Before any analysis, let's convert the target column - "income" into numerical classes where the label <=50K will become "0" and the label >50K will become "1". In order to achieve this numerical classes, we are using labelEncoder and fit transform in the desired numerical classes

In [4]:
```
from sklearn.preprocessing import LabelEncoder

labelEncoder = LabelEncoder()
census_data['Income'] = labelEncoder.fit_transform(census_data['Income'])
```

In [5]:
```
print(census_data.shape)
```

```
(32561, 15)
```

In [6]:
```
census_data.isnull().sum()
```

Out[6]:
```
Age                 0
Work Class          0
Final Weight        0
Education           0
Education Number    0
Marital Status      0
Occupation          0
Relationship        0
Race                0
Sex                 0
Capital Gain        0
Capital Loss        0
Hours per Week      0
Country             0
Income              0
dtype: int64
```

1. Exploratory Data Analysis and Data Processing

Using Exploratory Data Analysis (EDA) to understand the relationships for this classification problem where the labels are both numerical and categorical variables. Separation is achieved when there are distinctive feature values for each label category.

Let us take a look at the data and draw visualizations to understand it better. Let us take a look at the data collectively as a whole and then try to infer each column one by one. In this EDA Process, we need to Summarize variables, visualize distributions, and exploit relationships. Also, will Generate a few interesting questions about the data and explore them with some visualizations.

First of all, we need to identify any missing values in the censusdata set. To verify this, just to call censusdata. info () which tells us about the missing values if there are any discrepancies in the number of label entries.

In [40]:
```
census_data.info()
```

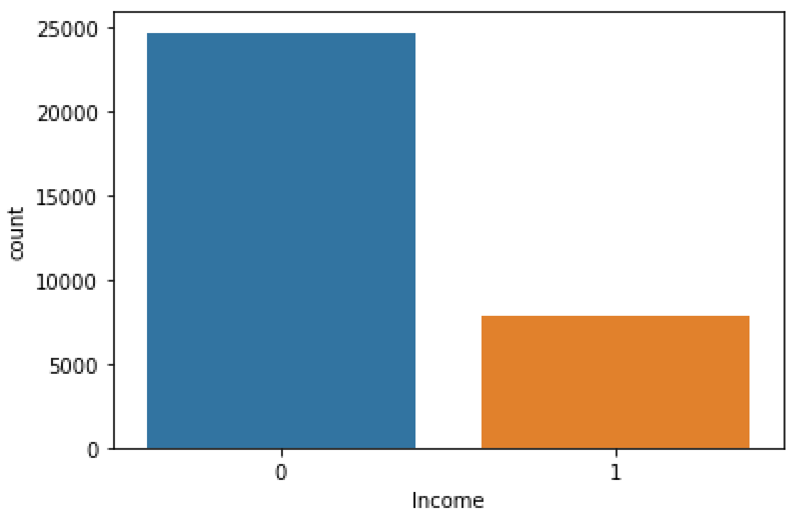
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30148 entries, 0 to 32560
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Age              30148 non-null  int64
 1   Work Class       30148 non-null  int8
 2   Education        30148 non-null  int8
 3   Education Number 30148 non-null  int64
 4   Marital Status   30148 non-null  int8
 5   Occupation       30148 non-null  int8
 6   Relationship     30148 non-null  int8
 7   Race             30148 non-null  int8
 8   Sex              30148 non-null  int8
 9   Hours per Week   30148 non-null  int64
 10  Country          30148 non-null  int8
 11  Income           30148 non-null  int32
 12  Net Capital      30148 non-null  int64
dtypes: int32(1), int64(4), int8(8)
memory usage: 1.5 MB
```

Here, the count of label entries for each of the 15 columns is 30148. no discrepancies found. Hence this is a good indication of "no missing values" in the data set.

Examine classes and class imbalance

In [8]:
```
sns.countplot(x='Income', data=census_data)
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x13413b36d90>
```



In this case, the label has significant class imbalance. Class imbalance means that there are unequal numbers of cases for the categories of the label. Class imbalance can seriously bias the training of classifier algorithms. It many cases, the imbalance leads to a higher error rate for the minority class. Most real-world classification problems have class imbalance, sometimes severe class imbalance, so it is important to test for this before training any model.

2. Research Methods

Here we calculated the sample correlation between at least one pair of variables (education number and sex) with a null hypothesis and calculated the p-value

In [49]:
```
census_data.describe()
```

Out[49]:

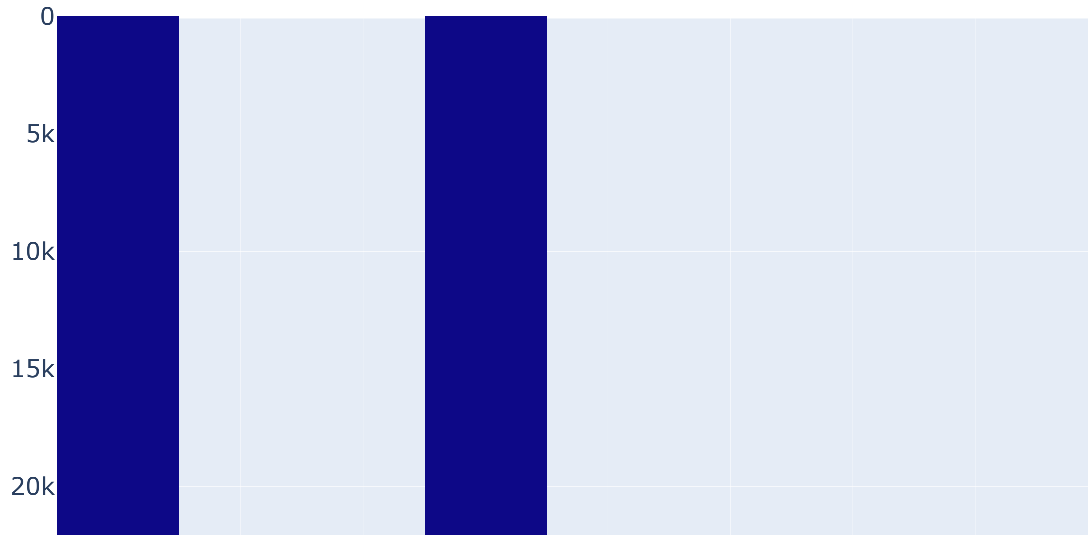
	Age	Work Class	Education	Education Number	Marital Status	Occupation	Re
count	30148.000000	30148.000000	30148.000000	30148.000000	30148.000000	30148.000000	301
mean	38.433561	2.197559	4.630921	10.121799	2.579972	5.960727	
std	13.128876	0.950623	2.337286	2.550246	1.498113	4.029692	
min	17.000000	0.000000	0.000000	1.000000	0.000000	0.000000	
25%	28.000000	2.000000	3.000000	9.000000	2.000000	2.000000	
50%	37.000000	2.000000	4.000000	10.000000	2.000000	6.000000	
75%	47.000000	2.000000	6.000000	13.000000	4.000000	9.000000	
max	90.000000	5.000000	8.000000	16.000000	6.000000	13.000000	

In [71]:
```
census_data.head()
```

Out[71]:

	Age	Work Class	Education	Education Number	Marital Status	Occupation	Relationship	Race	Sex	Hours per Week	Cou
0	39	5	2	13	4	0	1	1	1	40	
1	50	4	2	13	2	3	0	1	1	13	
2	38	2	4	9	0	5	1	1	1	40	
3	53	2	6	7	2	5	0	0	1	40	
4	28	2	2	13	2	9	5	0	0	40	

In [14]:
```
import plotly.express as px
fig = px.imshow(census_data)
fig.show()
```



From the above, census data interactive plot, it is quiet eveident that edcuation Number and sex have no correlation each other and therefore, we choose education number and sex to test hypothetically using t-test two samples.

In [100]:
```
census_data[['Education Number','Sex']].corr().round(3)
```

Out[100]:

	Education Number	Sex
Education Number	1.000	0.006
Sex	0.006	1.000

In [107]:
```
census_data[['Education Number','Sex']].count()
```

Out[107]:
```
Education Number    30148
Sex                 30148
dtype: int64
```

In [106]:
```
census_data[['Education Number','Sex']].groupby('Sex').count()
```

Out[106]:

Sex	Education Number
0	9777
1	20371

2.1 NULL HYPOTHESIS :

The education of people in the census data shall have no dependency upon their sex and have no correlation on each other.

$$H_0 : \mu_{EducationNumber,Sex=1} - \mu_{EducationNumber,Sex=0} = 0$$

In [104]:
```
census_data[['Education Number','Sex']].groupby('Sex').mean().round(3)
```

Out[104]:

Sex	Education Number
0	10.100
1	10.132

In [105]:
```
import pandas as pd
import numpy as np
import scipy.stats as ss
import math
import statsmodels.api as sm
import statsmodels.stats.weightstats as ws
from statsmodels.stats.power import tt_ind_solve_power


def t_test_two_samp(census_data, alpha, alternative='two-sided'):

    a = census_data[census_data.Sex == 1]['Education Number']
    b = census_data[census_data.Sex == 0]['Education Number']

    diff = a.mean() - b.mean()

    res = ss.ttest_ind(a, b)

    means = ws.CompareMeans(ws.DescrStatsW(a), ws.DescrStatsW(b))
    confint = means.tconfint_diff(alpha=alpha, alternative=alternative, usevar
='unequal')
    degfree = means.dof_satt()

    index = ['DegFreedom', 'Difference', 'Statistic', 'PValue', 'Low95CI', 'Hi
gh95CI']
    return pd.Series([degfree, diff, res[0], res[1], confint[0], confint[1]],
index = index)


test = t_test_two_samp(census_data, 0.05)
test
```

Out[105]:
```
DegFreedom    21247.962262
Difference        0.032973
Statistic         1.050889
PValue            0.293318
Low95CI          -0.026267
High95CI          0.092213
dtype: float64
```

Q# 2.2 p-value

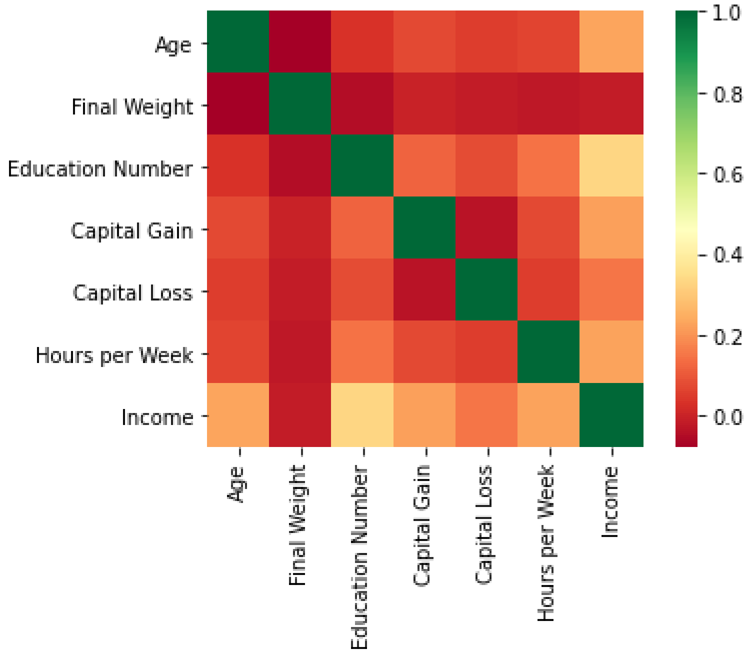
Notice that the p-value is 0.293318 (PValue from above Table). This is obviously not less than .05 ($p < .05$) so we can accept the null hypothesis and conclude that the education of people in the census data will not have any dependence on thier sex which was not a statistical fluke but likely a real trend in the population.

3. Data Cleaning and Preparation

Here we applied the appropriate preprocessing steps, such as removing duplicates, missing values, outliers, and scaling data as appropriate.

In [9]:
```
sns.heatmap(census_data.corr(), square=True, cmap='RdYlGn')
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x13414a3b460>



In the above heatmap, it is quiet evident that "income" has no correlation with "Final Weight" so that we can remove this column for the income prediction

In [10]:
```
census_data.drop(['Final Weight'], axis = 1, inplace = True)
```

Age

Here, age can be separated into the following classes.

0-25: Young

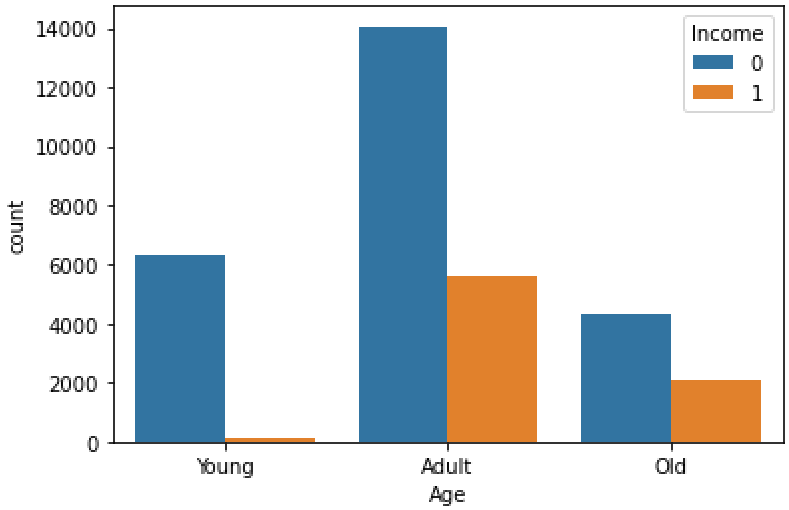
25-50: Adult

50-100: Old

In [29]:
```
census_data['Age'] = pd.cut(census_data['Age'], bins = [0, 25, 50, 100], label
s = ['Young', 'Adult', 'Old'])
```

In [30]:
```
sns.countplot(x = 'Age', hue = 'Income', data = census_data)
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x17124603910>



It appears that there are relatively less "Young" people who have an income more than "$50K". Young people are the begining of their career and most of them may have entry salaries.

4. Feature Engineering

Here we created new features or transform existing ones to improve performance.

Capital Gain and Capital Loss

Rather than having both Capital Gain and Capital Loss, let us use their absolute difference as " Net Capital" that would be more relevant and convenient.

In [11]:
```
census_data['Net Capital'] = abs(census_data['Capital Gain'] - census_data['Ca
pital Loss'])
census_data.drop(['Capital Gain'], axis = 1, inplace = True)
census_data.drop(['Capital Loss'], axis = 1, inplace = True)
census_data.head()
```

Out[11]:

	Age	Work Class	Education	Education Number	Marital Status	Occupation	Relationship	Race	Sex	Hours per Week
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	40
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	13
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	40
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	40
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	40

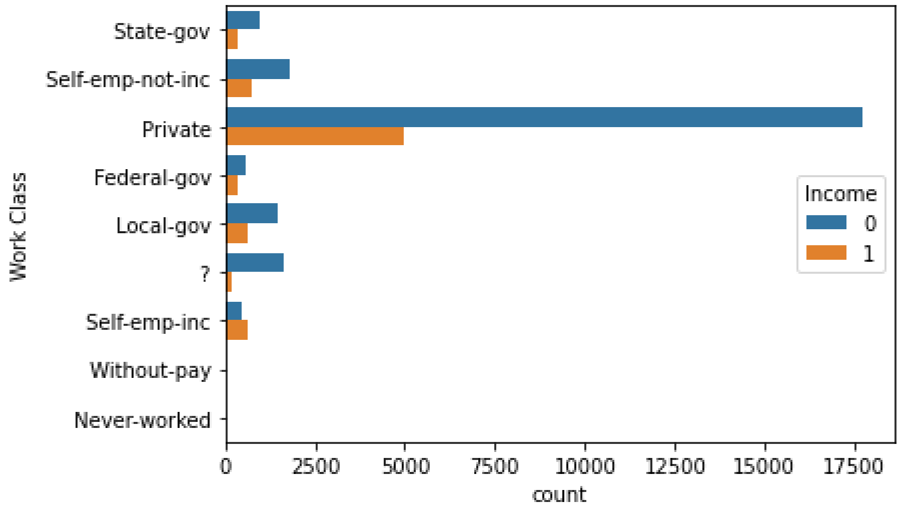
On taking a look at the result, we can see that the lable for "Zero", there are more people with Income less than 50K (income - 0) and for lable "Non-Zero" there are more people with Income greater than 50K (Income - 1).

Work Class

Obviously the work classes have direct correlation to the income of people . Based on the work classes, it is easy to predict the salary range, thereby the income class.

In [12]:
```
sns.countplot(y = 'Work Class', hue = 'Income', data = census_data)
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x134149c83d0>



From the above plot, we can see that there are Work Class values shown as "?" which could be error data. As it is very less in the count, we can simply remove these records. Also, the two values "Without-pay" and "Never-worked" are negligible to count and hence it is safe to drop them too.

In [13]:
```
census_data = census_data.drop(census_data[census_data['Work Class'] == ' ?'].index)
census_data = census_data.drop(census_data[census_data['Work Class'] == ' Without-pay'].index)
census_data = census_data.drop(census_data[census_data['Work Class'] == ' Never-worked'].index)
```

Education and Education Number

It is obvious that Education Number and Education are related to each other. Education Number is representing the corresponding education. Also, we can combine all information from Preschool to 12th and they can be considered of one class as "Preschool" who have no college/university level education.

In [15]:
```
census_data['Education'].replace([' 11th', ' 9th', ' 7th-8th', ' 5th-6th', ' 1
0th', ' 1st-4th', ' Preschool', ' 12th'],
                                 ' Preschool', inplace = True)
census_data['Education'].value_counts()
```

Out[15]:
```
 HS-grad         9959
 Some-college    6772
 Bachelors       5182
 Preschool       3820
 Masters         1675
 Assoc-voc       1321
 Assoc-acdm      1019
 Prof-school      558
 Doctorate        398
Name: Education, dtype: int64
```

Race

In the dataset, regarding race which includes majority of information about "White" race while all other races are mentioned very few cases only. Hence, we shall combine all other races data into one class as "Other”.

In [16]:
```
census_data['Race'].unique()
census_data['Race'].replace([' Black', ' Asian-Pac-Islander', ' Amer-Indian-Es
kimo', ' Other'],' Other', inplace = True)
```

Country

In [17]:
```
country_count = census_data['Country'].value_counts()
country_count
```

Out[17]:
```
United-States                 27491
Mexico                          610
?                               556
Philippines                     187
Germany                         128
Puerto-Rico                     109
Canada                          107
El-Salvador                     100
India                           100
Cuba                             92
England                          86
Jamaica                          80
South                            71
China                            68
Italy                            68
Dominican-Republic               67
Vietnam                          64
Guatemala                        63
Japan                            59
Columbia                         56
Poland                           56
Iran                             42
Haiti                            42
Taiwan                           42
Portugal                         34
Nicaragua                        33
Peru                             30
Greece                           29
Ecuador                          27
France                           27
Ireland                          24
Hong                             19
Trinadad&Tobago                  18
Cambodia                         18
Laos                             17
Thailand                         17
Yugoslavia                       16
Outlying-US(Guam-USVI-etc)       14
Hungary                          13
Honduras                         12
Scotland                         11
Holand-Netherlands                1
Name: Country, dtype: int64
```

From the above country count table, there are some missing values in Country column denoted by “?”. As they are very cases only, these rows are dropped from the data set. The majority of adults are from United States. Thus, we can distribute the column with values as either “United States” or “Other”.

In [18]:
```
census_data = census_data.drop(census_data[census_data['Country'] == ' ?'].ind
ex)
countries = np.array(census_data['Country'].unique())
countries = np.delete(countries, 0)
census_data['Country'].replace(countries, 'Other', inplace = True)
census_data.head()
```

Out[18]:

	Age	Work Class	Education	Education Number	Marital Status	Occupation	Relationship	Race	Sex	Hours per Week
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	40
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	13
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	40
3	53	Private	Preschool	7	Married-civ-spouse	Handlers-cleaners	Husband	Other	Male	40
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Other	Female	40

5. Model Selection

Here we tred the following models to show the evaluation process. It is clearly indicated 1) metrics used in the model and 2) the performance of each model. Imbalance in the data has been addressed as well by using appropriate train/test data split.

Machine Learning Models

Applications of Classification

A classifier is a machine learning model that separates the label into categories or classes. In other words, classification models are supervised machine learning models which predict a categorical label.

Here we considered the following

To prepare data for classification models using scikit-learn and imblance-learn packages. Constructing a classification model using scikit-learn and imblance-learn packages. Evaluating the performance of the classification models. Using techniques such as reweighting the labels and changing the decision threshold to change the trade-off between false positive and false negative error rate

Data Manipulation

With the data prepared, to create the numpy arrays required for the scikit-learn model.

To create the numpy feature array or model matrix. We need to follow the following steps

Seggregate into categorical columns and numeric columns Transform the integer coded variables to dummy variables. Encode the categorical string variables as integers. Split the cases into training and test data sets. If machine learning models are tested on the training data, the results will be both biased and overly optimistic. Numeric features must be rescaled so they have a similar range of values. Rescaling prevents features from having an undue influence on model training simply because then have a larger range of numeric variables. Append each dummy coded categorical variable to the model matrix. Execute the code in the cell below to perform this processing and examine the results

In [19]:
```
cat_columns = ['Work Class', 'Marital Status', 'Occupation', 'Relationship',
'Race', 'Sex', 'Country']
num_columns = ['Age', 'Education Number', 'Net Capital']
```

In [20]:
```
for col in census_data.columns:
    if census_data[col].dtype==object:
            census_data[col]=census_data[col].astype('category')
            census_data[col]=census_data[col].cat.codes
```

OneHotEncoder

Encode categorical features as a one-hot numeric array. This encoding is needed for feeding categorical data to many scikit-learn estimators, notably linear models and SVMs with the standard kernels.

In [21]:
```
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
df_cat = census_data[cat_columns]
encoded = ohe.fit_transform(df_cat)
ohe_df = pd.DataFrame(encoded.todense(), columns=ohe.get_feature_names())
```

In [22]:
```
df_num = census_data[num_columns].reset_index(drop=True)
```

In [23]:
```
y = census_data['Income']
```

In [24]:
```
X = pd.concat([df_num, ohe_df], axis = 1)
y = y
```

In [25]:
```
X.shape, y.shape
```

Out[25]:
```
((30148, 42), (30148,))
```

Next, let us split the dataset into the training and testing data using train_test_split.

In [29]:
```
from sklearn import model_selection
from sklearn.model_selection import train_test_split
seed = 7
test_size = 0.2
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test
_size=0.2, random_state=seed)
print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

```
Number transactions X_train dataset:  (24118, 42)
Number transactions y_train dataset:  (24118,)
Number transactions X_test dataset:  (6030, 42)
Number transactions y_test dataset:  (6030,)
```

Score and evaluate the classification model

Given the results of the test data, here, in order to quantify the performance of the model, we used confusion matrix to evaluate the performance of the machine leaning model classifiers. The confusion matrix lays out the correctly and incorrectly classified cases in a tabular format.

Confusion matrix

Here the four elements in the matrix are defined as:
True Positive or **TP** are cases with positive labels which have been correctly classified as positive.
True Negative or **TN** are cases with negative labels which have been correctly classified as negative.
False Positive or **FP** are cases with negative labels which have been incorrectly classified as positive.
False Negative or **FN** are cases with positive labels which have been incorrectly classified as negative.

When creating a confusion matrix it is important to understand and maintain a convention for which differentiating positive and negative label values. The usual convention is to call the 1 case positive and the 0 case negative.

Accuracy : Accuracy is a simple and often misused metric. In simple terms, accuracy is the fraction of cases correctly classified. For a two-class classifier accuracy is written as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision :Precision is the fraction of correctly classified label cases out of all cases classified with that label value. We can express precision by the following relationship:

$$Precision = \frac{M_{i,i}}{\sum_j M_{i,j}}$$

Recall : Recall is the fraction of cases of a label value correctly classified out of all cases that actually have that label value. We can express recall by the following relationship:

$$Recall = \frac{M_{i,i}}{\sum_i M_{i,j}}$$

F1 :The F1 statistic is weighted average of precision and recall. We can express F1 by the following relationship:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

ROC and **AUC** : The receiver operating characteristic or ROC is a curve that displays the relationship between the true positive rate on the vertical axis and false positive rate on the horizontal axis. The ROC curve shows the tradeoff between true positive rate and false positive rate.

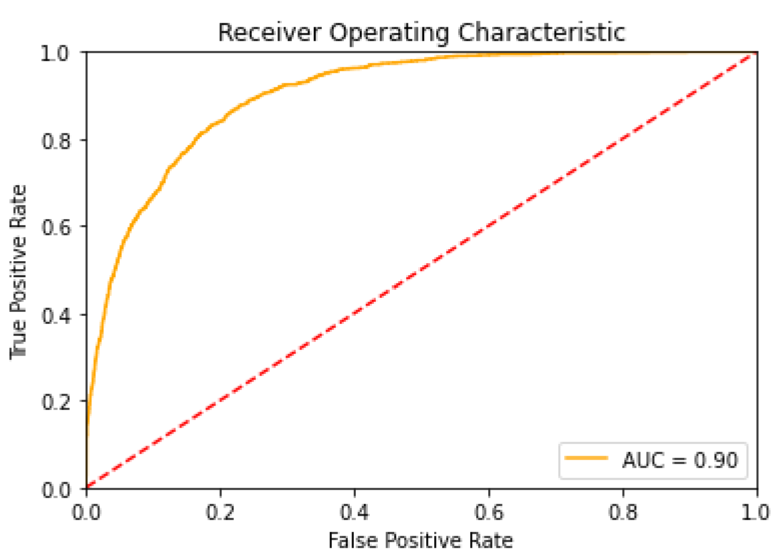
The AUC is the area or integral under the ROC curve. The overall performance of the classifier is measured by the area under the curve or AUC. The higher the AUC the lower the increase in false positive rate required to achieve a required true positive rate. For an ideal classifier the AUC is 1.0. A true positive rate is achieved with a 0 false positive rate. This behavior means that AUC is useful for comparing classifiers. The classifier with higher AUC is generally the better one.

In [69]:
```
from sklearn import metrics as sklm
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
```

Classify The Model Using LogisticRegression Pipeline

In [138]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression

pipe_clf = Pipeline([('scaler', StandardScaler()), ('clf', LogisticRegression
())])
pipe_clf.fit(X_train, y_train)

pipe_clf.score(X_test, y_test)

y_pred = pipe_clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

pred_test =pipe_clf.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

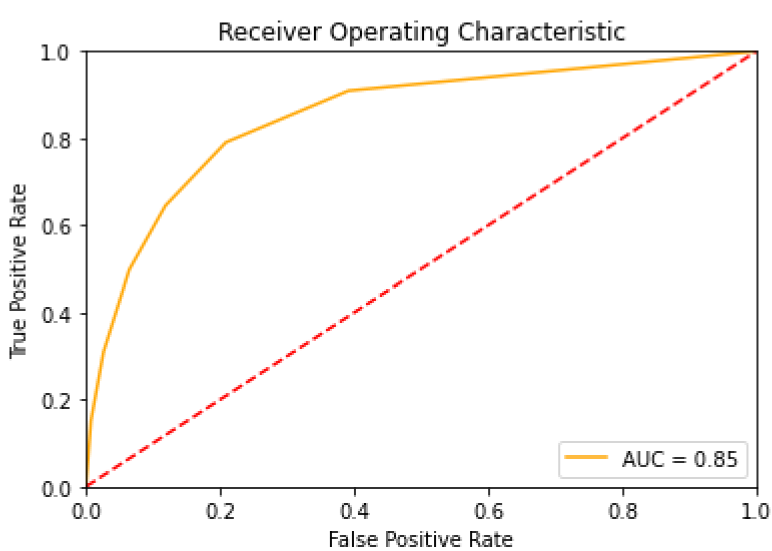
```
[[4236  320]
 [ 581  893]]
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      4556
           1       0.74      0.61      0.66      1474

    accuracy                           0.85      6030
   macro avg       0.81      0.77      0.78      6030
weighted avg       0.84      0.85      0.85      6030
```



Classify The Model Using KNeighborsClassifier Pipeline

In [139]:
```
from sklearn.neighbors import KNeighborsClassifier

pipe_knn = Pipeline([('scaler', StandardScaler()), ('knn', KNeighborsClassifie
r(n_neighbors = 6))])
pipe_knn.fit(X_train, y_train)

pipe_knn.score(X_test, y_test)

y_pred = pipe_knn.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

pred_test =pipe_knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

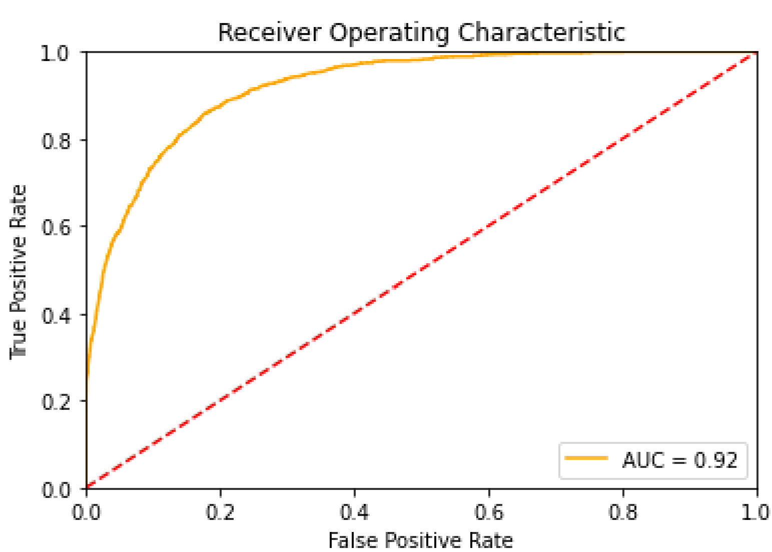
```
[[4262  294]
 [ 740  734]]
              precision    recall  f1-score   support

           0       0.85      0.94      0.89      4556
           1       0.71      0.50      0.59      1474

    accuracy                           0.83      6030
   macro avg       0.78      0.72      0.74      6030
weighted avg       0.82      0.83      0.82      6030
```



Classify The Model Using GradientBoostingClassifier Pipeline

In [140]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import GradientBoostingClassifier

pipe_gb = Pipeline([('scaler', StandardScaler()), ('gb', GradientBoostingClass
ifier(random_state = 0))])
pipe_gb.fit(X_train, y_train)

pipe_gb.score(X_test, y_test)

y_pred = pipe_gb.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

pred_test =pipe_gb.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

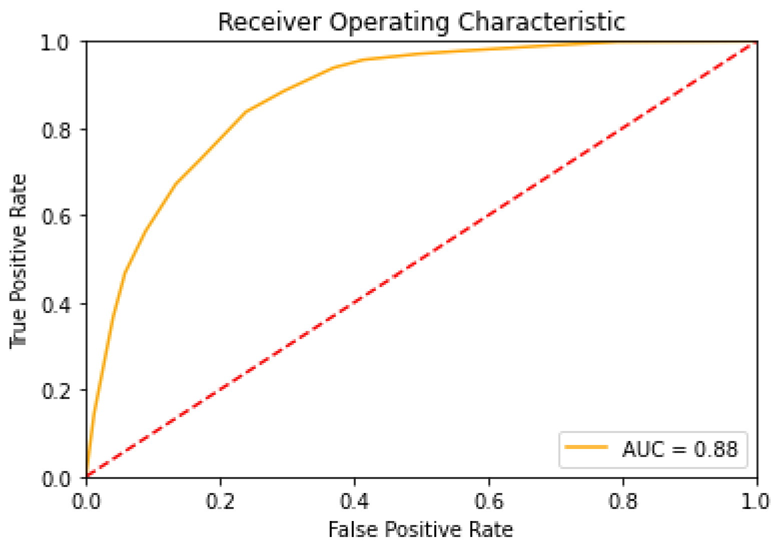
```
[[4303  253]
 [ 574  900]]
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      4556
           1       0.78      0.61      0.69      1474

    accuracy                           0.86      6030
   macro avg       0.83      0.78      0.80      6030
weighted avg       0.86      0.86      0.86      6030
```



Classify The Model Using DecisionTreeClassifier with GridSearchCV

In [58]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import metrics as sklm
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score

SEED = 1

dt = DecisionTreeClassifier(random_state=SEED)
params_dt = {
             'max_depth':[3, 4, 5, 6],
             'min_samples_leaf': [0.04, 0.06, 0.08],
             'max_features': [0.2, 0.4, 0.6, 0.8]
            }

grid_dt = GridSearchCV(estimator = dt, param_grid=params_dt,scoring='roc_auc',
cv=10, n_jobs=-1)
model_grid_dt = grid_dt.fit(X_train, y_train)

best_hyperparams = grid_dt.best_params_
print('Best Hyperparameters:\n', best_hyperparams)

y_pred = grid_dt.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

pred_test =grid_dt.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
Best Hyperparameters:
 {'max_depth': 6, 'max_features': 0.8, 'min_samples_leaf': 0.04}
[[4292  264]
 [ 787  687]]
              precision    recall  f1-score   support

           0       0.85      0.94      0.89      4556
           1       0.72      0.47      0.57      1474

    accuracy                           0.83      6030
   macro avg       0.78      0.70      0.73      6030
weighted avg       0.82      0.83      0.81      6030
```



Performance Optimization

In the above, we used regularization, hyperparameter tuning and now we are using other techniques such as SMOTE to further optimize model and/or help select the best model.

SMOTE - Synthetic Minority Over-sampling Technique

To deal with data imbalance we use SMOTE - Synthetic Minority Over-sampling Technique. SMOTE creates synthetic (not duplicate) samples of the minority class. Hence making the minority class equal to the majority class. SMOTE does this by selecting similar records and altering that record one column at a time by a random amount within the difference to the neighbouring records. SMOTE is imported from imbalance learn over_sampling

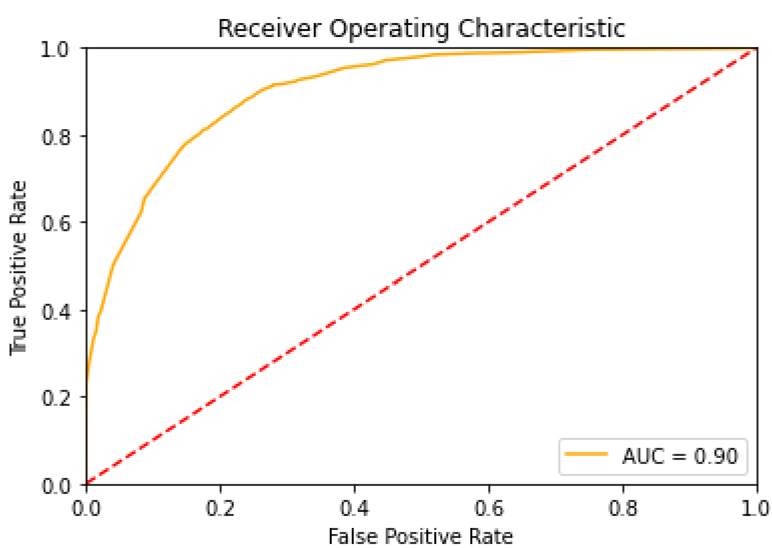
In [63]:
```
from imblearn.under_sampling import NearMiss
from imblearn.pipeline import make_pipeline
from imblearn.metrics import classification_report_imbalanced
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE


print("Before OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train==0
)))

sm = SMOTE(random_state=2)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape
))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.sha
pe))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res==1
)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res==0
)))
```

```
Before OverSampling, counts of label '1': 6034
Before OverSampling, counts of label '0': 18084

After OverSampling, the shape of train_X: (36168, 42)
After OverSampling, the shape of train_y: (36168,)

After OverSampling, counts of label '1': 18084
After OverSampling, counts of label '0': 18084
```

Classify the model using DecisionTreeClassifier with SMOTE

In [64]:
```
clf = DecisionTreeClassifier(max_depth = 10,min_samples_split = 500)
model_clf = clf.fit(X_train_res,y_train_res)
y_pred = clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

pred_test =clf.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

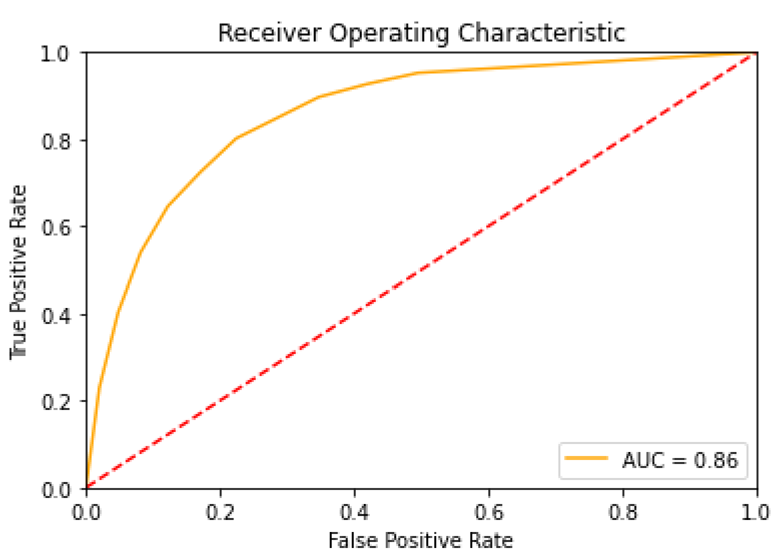
```
[[3763  793]
 [ 280 1194]]
              precision    recall  f1-score   support

           0       0.93      0.83      0.88      4556
           1       0.60      0.81      0.69      1474

    accuracy                           0.82      6030
   macro avg       0.77      0.82      0.78      6030
weighted avg       0.85      0.82      0.83      6030
```



Classify the model using KNeighborsClassifier with SMOTE

In [67]:
```
knn = KNeighborsClassifier(n_neighbors = 10)
knn.fit(X_train_res, y_train_res)
y_pred = knn.predict(X_test)
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)

print(confusion_matrix)
print(classification_report(y_test, y_pred))


pred_test2 =knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test2)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
[[3777  779]
 [ 406 1068]]
              precision    recall  f1-score   support

           0       0.90      0.83      0.86      4556
           1       0.58      0.72      0.64      1474

    accuracy                           0.80      6030
   macro avg       0.74      0.78      0.75      6030
weighted avg       0.82      0.80      0.81      6030
```



Various Classification models Algorithm Comparison

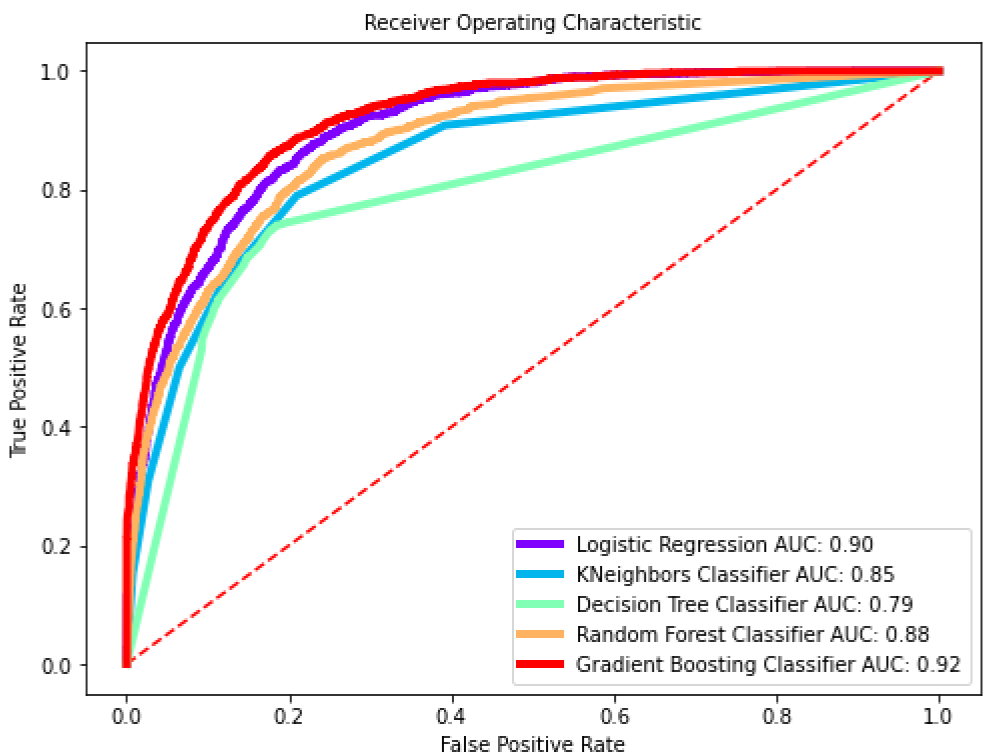
In [130]:
```
from sklearn.metrics import f1_score, accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifie
r
```

In [131]:
```
classifiers = [LogisticRegression(),
               KNeighborsClassifier(n_neighbors = 6),
               DecisionTreeClassifier(random_state = 1),
               RandomForestClassifier(n_estimators = 100, random_state = 0),
               GradientBoostingClassifier(random_state = 0)]
classifier_names = ["Logistic Regression",
                    "KNeighbors Classifier",
                    "Decision Tree Classifier",
                    "Random Forest Classifier",
                    "Gradient Boosting Classifier"]
accuracies = []
```

In [132]:
```
for i in range(len(classifiers)):
    classifier = classifiers[i]
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
```

In [136]:
```
from sklearn.metrics import roc_curve, auc
from matplotlib.cm import rainbow

plt.figure(figsize = (8, 6))
plt.plot([0,1], [0,1], 'r--')
colors = rainbow(np.linspace(0, 1, len(classifiers)))

for i in range(len(classifiers)):
    classifier = classifiers[i]
    probs = classifier.predict_proba(X_test)

    probs = probs[:, 1]
    fpr, tpr, thresholds = roc_curve(y_test, probs)
    roc_auc = auc(fpr, tpr)
    label = classifier_names[i] + ' AUC:' + ' {0:.2f}'.format(roc_auc)
    plt.plot(fpr, tpr, c = colors[i], label = label, linewidth = 4)

plt.xlabel('False Positive Rate', fontsize = 10)
plt.ylabel('True Positive Rate', fontsize = 10)
plt.title('Receiver Operating Characteristic', fontsize = 10)
plt.legend(loc = 'lower right', fontsize = 10)
```

Out[136]: <matplotlib.legend.Legend at 0x1342a246f40>



Making Prediction using Best Model Algorithm - GradientBoostingClassifier

In [141]:
```
def score_model(probs, threshold):
    return np.array([0 if x > threshold else 1 for x in probs[:,1]])
threshold = 0.51
probabilities = pipe_gb.predict_proba(X_test)
scores = score_model(probabilities, threshold)
```

In [161]:
```
import collections
#pd.set_option("max_columns", None)
scores=collections.Counter(scores)
print(scores)
```

```
Counter({1: 4906, 0: 1124})
```

In [143]:
```
np.savetxt('final_answers_1.csv', scores, delimiter=',',fmt='%i')
```

Summary:

1. Loaded the data using pandas and identified the target variable “Income” and converted the target column - "Income" into numerical classes where the label <=50K will become "0" and the label >50K will become "1"
2. Examined classes and class imbalance using count plot.
3. Applied statistical research methods to analyze the correlation between education number and sex. Using interactive census data plot, identified the two variables, education number and sex.
4. Used appropriate preprocessing steps, such as removing duplicates, missing values, outliers, and scaling the data.
5. Identified the most correlated variables in the census data using a heatmap. "income" has no correlation with "Final Weight" so that can be removed from the income prediction.
6. Classified Age in to “Young”, “Adult” and “Old” classes. By using a count plot, identified that relatives a smaller number of "Young" people who have an income more than "$50K.
7. Using appropriate Feature Engineering techniques, combined Capital Gain and Capital Loss by their absolute difference into a single feature as " Net Capital" that would be more relevant and convenient. Then, dropped Capital Gain and Capital Loss columns.
8. From the Work Class count plot, there are Work Class values shown as "?" which could be error data. As it is very less in the count, these records were removed. Also, the two values "Without-pay" and "Never-worked" are negligible to count and hence it is dropped from the data set.
9. Education Number and Education are related to each other. Education Number is representing the corresponding education. Also, we can combine all information from Preschool to 12th and they can be considered of one class as "Preschool" who have no college/university level education.
10. In the dataset, regarding race which includes majority of information about "White" race while all other races are mentioned very few cases only. Hence, we shall combine all other races data into one class as "Other”.
11. From the country count table, there are some missing values in Country column denoted by “?”. As they are very cases only, these rows are dropped from the data set. The majority of adults are from United States. Thus, we can distribute the column with values as either “United States” or “Other”.
12. For Machine Learning Models, it was used AUC as metric because this is highly imbalanced dataset.
13. Segregated Categorical Columns and Numerical Columns
14. Data Manipulation: Applied OneHotEncoder to encode the all categorical features and Created Numerical Column from the census_data frame
15. Spitted the whole data as train, test using model_selection.train_test_split with seed = 7 and test size = 0.2
16. Applied Machine Learning algorithms such as

    1) LogisticRegression Pipeline with Accuracy = 0.85 and AUC = 0.90

    2) KNeighborsClassifier Pipeline with Accuracy = 0.83 and AUC = 0.85

    3) GradientBoostingClassifier Pipeline with Accuracy = 0.86 and AUC = 0.92

    4) DecisionTreeClassifier with GridSearchCV with Accuracy = 0.83 and AUC = 0.88

    5) DecisionTreeClassifier with SMOTE with Accuracy = 0.82 and AUC = 0.90

    6) KNeighborsClassifier with SMOTE with Accuracy = 0.80 and AUC = 0.86
17. Created a rainbow of Comparison of ROC Curves of the above Classification models Algorithms
18. Best Model Algorithm Evaluated as - GradientBoostingClassifier with Accuracy = 0.86 and AUC = 0.92
19. Created a CSV file for Prediction using the Best Model Algorithm - GradientBoostingClassifier Pipeline

```
In [ ]:
```