# Census Income Project

Practice problems or data science projects are one of the best ways to learn data science. **"You don’t learn data science until you start working on problems yourself."**



**Problem Statement :**

This data was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics). The data can be downloaded [here](https://github.com/SheikhSAli/Machine-Learning-Blogs/blob/main/Census_raw_data.txt)

**Hypothesis**

A set of reasonably clean records was extracted using the following conditions: (**AGE > 16**) :- Age of the observed individual should be greater than 16 yrs. (**AFNLWGT > 1**) :- Final Weight should br greater than 1. (**HRSWK > 0**):- Hours per week should be greater than 0. The prediction task is to determine whether a person makes over $50K a year.

#### **Data Exploration**

We’ll be performing some basic data exploration here and come up with some inferences about the data. We’ll try to figure out some irregularities and address them in the next section. If you are new to this domain, please refer our [Data Exploration Guide.](https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/)

**Age** :- Age of the observed individual.

**Work\_Class** :-This feature has values associated with the working class of an individual , It has 8 categories with following value counts.

**Final\_Weight** :- Description of fnlwgt (final weight) The weights on the Current Population Survey (CPS) files are controlled to independent estimates of the civilian non-institutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau. We use 3 sets of controls. These are: A single cell estimate of the population 16+ for each state. Controls for Hispanic Origin by age and sex. Controls by Race, age and sex. We use all three sets of controls in our weighting program and "rake" through them 6 times so that by the end we come back to all the controls we used. The term estimate refers to population totals derived from CPS by creating "weighted tallies" of any specified socio-economic characteristics of the population. People with similar demographic characteristics should have similar weights. There is one

important caveat to remember about this statement. That is that since the CPS sample is actually a collection of 51 state samples, each with its own probability of selection, the statement only applies within state.

**Education** :- This feature has the level of education of an individual. This feature has 9 different value counts .

**Education Number** :- This feature is almost similar to Education , difference is this feature has numbers given for any specified education.

**Occupation** :- This feature has the details of the occupation of an individual.

**Relationship** :- This feature describes the relationship of an individual with the household . This feature has 6 different criterian.

**Race** :- This feature has the race of an individual.

**Sex** :- This feature has the corresponding gender of an individual.

**Capital Gain** :- Capital gain is the profit one earns on the sale of an asset like stocks, bonds or real estate. It results in capital gain when the selling price of an asset exceeds its purchase price. It is the difference between the selling price (higher) and cost price (lower) of the asset.

**Capital Loss** :- Capital loss arises when the cost price is higher than the selling price.

**Hours per Week** :- The time in hour an individual spend a week in working.

**Country** :- Individuals belonging to a countriey.

**Target Variable** **Income** - By considering all the above factors we need to predict the Income of an individual whether its <=50k or >50k. So this is a classification problem .

To read more about the data , value counts and explore more visit this [Census\_Data\_Description](https://github.com/SheikhSAli/Machine-Learning-Blogs/blob/main/Census_Data_Description.xlsx).

#### The prediction task is to determine whether a person makes over $50K a year.

### Importing Libraries

Here we are importing some necessary libraries needed for our classification problem.

To learn about the libraries kindly follow the link. [Numpy](https://numpy.org/) , [Pandas](https://pandas.pydata.org/) , [Matplotlib](https://matplotlib.org/stable/index.html) , [Seaborn](https://seaborn.pydata.org/)

import numpy as np   
import pandas as pd  
import matplotlib.pyplot as plt  
%matplotlib inline  
plt.style.use("ggplot")  
from matplotlib.cm import rainbow  
import seaborn as sns  
from sklearn.preprocessing import LabelEncoder  
from sklearn.preprocessing import PowerTransformer  
from sklearn.preprocessing import StandardScaler  
from scipy.stats import zscore  
from matplotlib import rcParams  
from sklearn.metrics import plot\_confusion\_matrix  
from sklearn.naive\_bayes import GaussianNB  
from sklearn import linear\_model

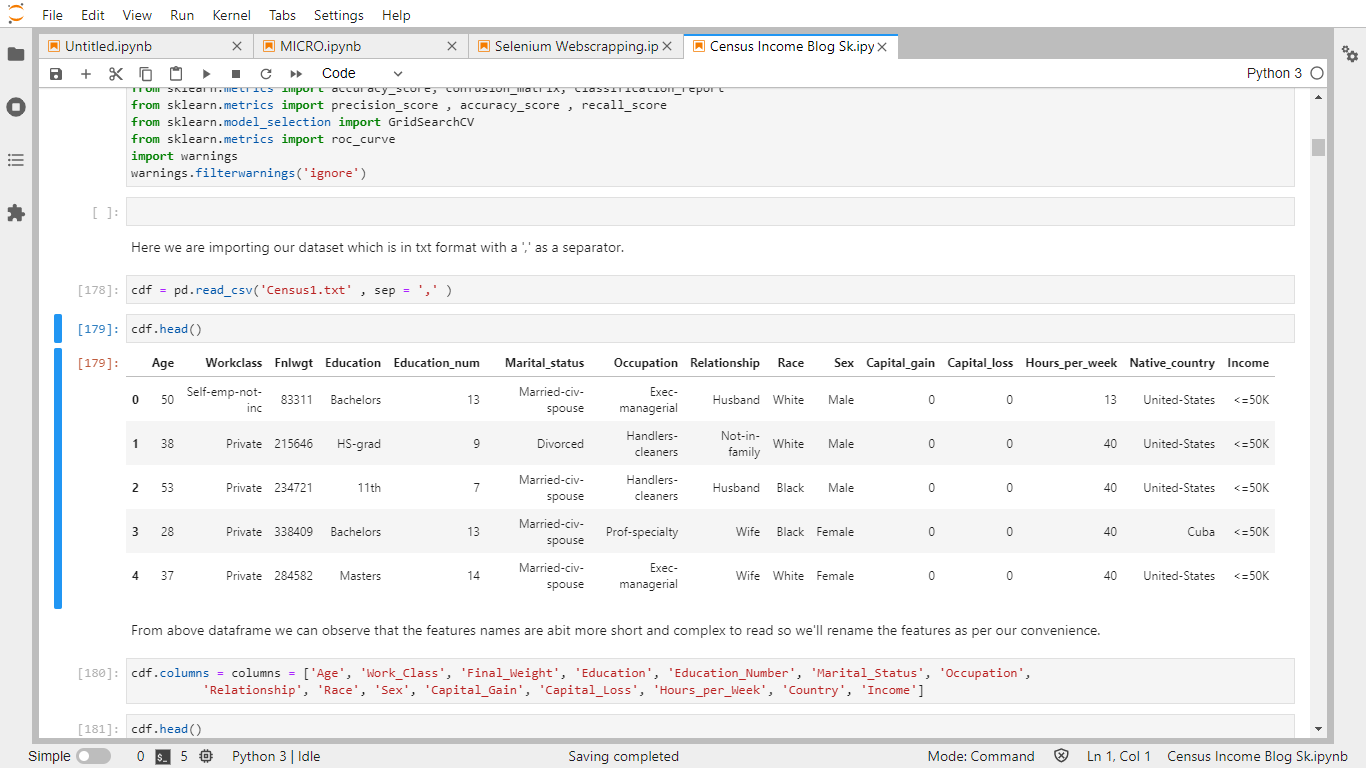
from sklearn.metrics import fbeta\_score  
from sklearn.metrics import accuracy\_score

from sklearn.metrics import f1\_score  
from sklearn.metrics import roc\_curve, auc  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.svm import SVC  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import cross\_val\_predict  
from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report  
from sklearn.metrics import precision\_score , accuracy\_score , recall\_score  
from sklearn.model\_selection import GridSearchCV  
from sklearn.metrics import roc\_curve  
import warnings   
warnings.filterwarnings('ignore')

Here we are importing our dataset which is in txt format with a ',' as a separator.

cdf = pd.read\_csv('Census1.txt' , sep = ',' )

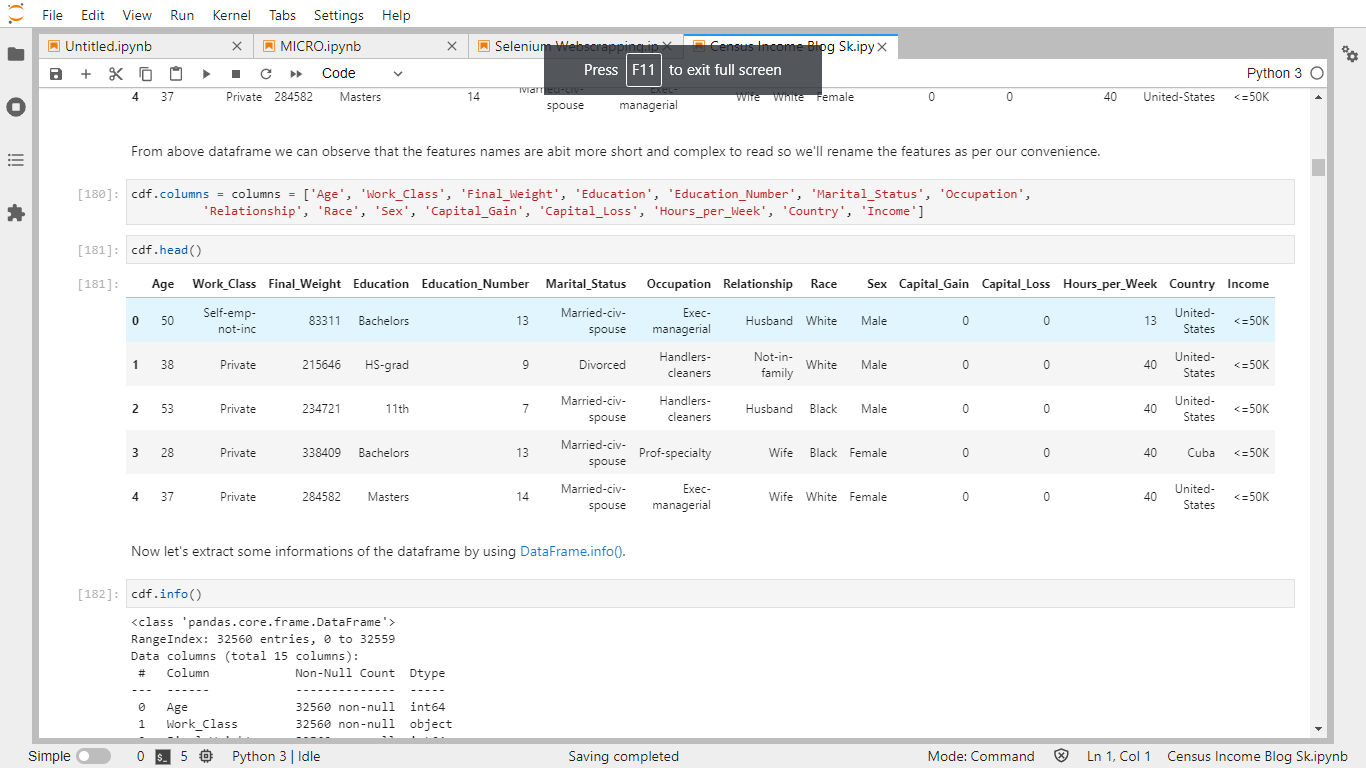
cdf.head()



From above dataframe we can observe that the features names are abit more short and complex to read so we'll rename the features as per our convenience.

cdf.columns = columns = ['Age', 'Work\_Class', 'Final\_Weight', 'Education', 'Education\_Number', 'Marital\_Status','Occupation', 'Relationship', 'Race', 'Sex', 'Capital\_Gain', 'Capital\_Loss', 'Hours\_per\_Week', 'Country', 'Income']

cdf.head()



Now let's extract some informations of the dataframe by using [DataFrame.info()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.info.html).

cdf.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 Work\_Class 32560 non-null object  
 2 Final\_Weight 32560 non-null int64   
 3 Education 32560 non-null object  
 4 Education\_Number 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 Country 32560 non-null object  
 14 Income 32560 non-null object  
dtypes: int64(6), object(9)  
memory usage: 3.7+ MB

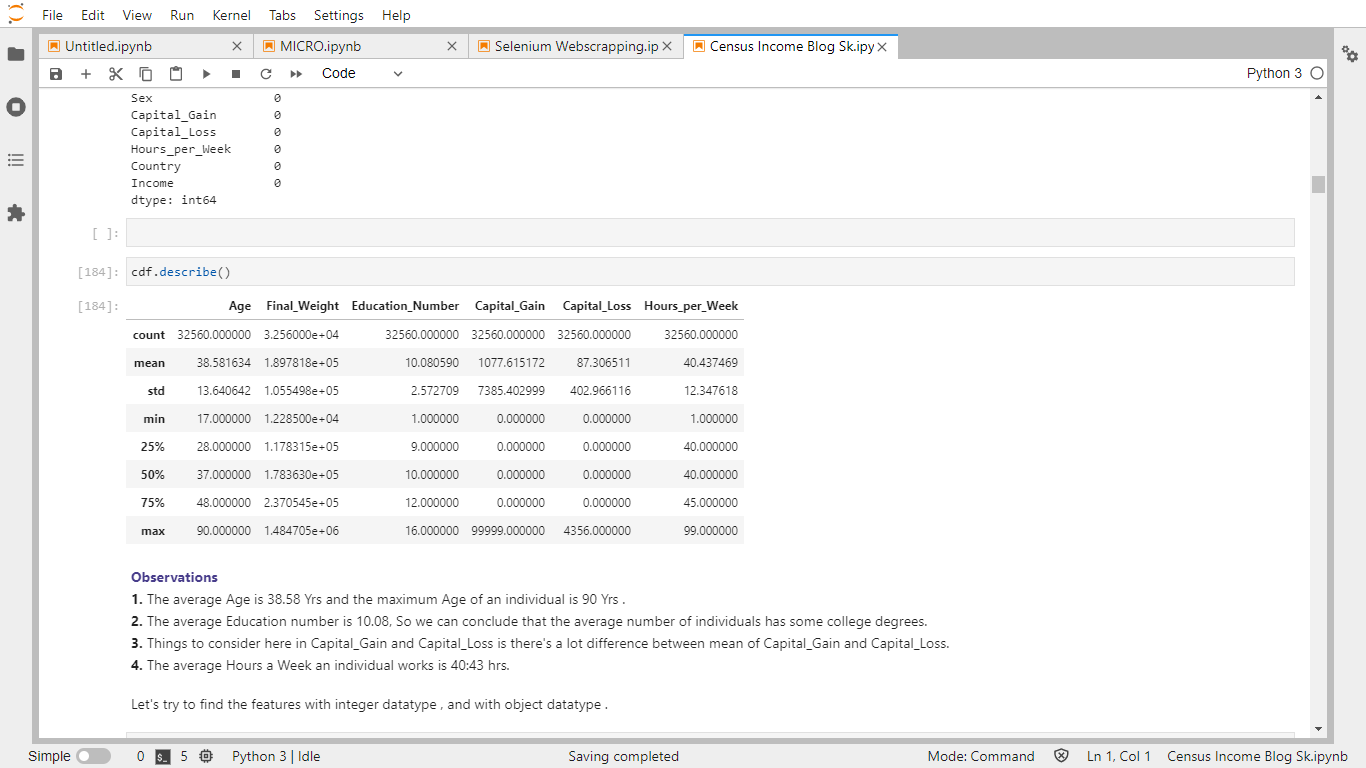
From above description we can observe that there is -A total of 15 columns are there. -A total of 32560 entries are there. -There is no Null entry present. -A total of 6 integer type and 9 object type feature is there. -The total memory taken by this DataFrame is more than 3.7 MB.

### Exploratory Data Analysis (EDA)

cdf.isnull().sum()

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

cdf.describe()



**Observations** :-

**1.** The average Age is 38.58 Yrs and the maximum Age of an individual is 90 Yrs .

**2.** The average Education number is 10.08, So we can conclude that the average number of individuals has some college degrees.

**3.** Things to consider here in Capital\_Gain and Capital\_Loss is there's a lot difference between mean of Capital\_Gain and Capital\_Loss.

**4.** The average Hours a Week an individual works is 40:43 hrs.

Let's try to find the features with integer datatype , and with object datatype .

int\_type = [feature for feature in cdf.columns if cdf[feature].dtypes !='O']  
  
print("Below are the features with Integer datatype and the total count is :" , len(int\_type))  
print(int\_type)

Below are the features with Integer datatype and the total count is : 6  
['Age', 'Final\_Weight', 'Education\_Number', 'Capital\_Gain', 'Capital\_Loss', 'Hours\_per\_Week']

object\_type = [feature for feature in cdf.columns if cdf[feature].dtypes =='O']  
  
print("Below are the features with Object datatype and the total count is :" , len(object\_type))  
print(object\_type)

Below are the features with Object datatype and the total count is : 9  
['Work\_Class', 'Education', 'Marital\_Status', 'Occupation', 'Relationship', 'Race', 'Sex', 'Country', 'Income']

Above is the list of all features with their respective datatypes.

#### **Age**

cdf.Age.value\_counts()

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

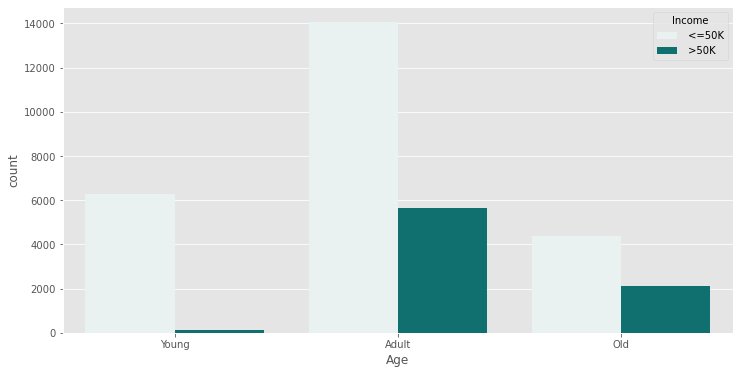
From above we can observe that there are a lot of values of Age feature . So Here, we'll group the ages into separate bins as mentioned below.

0-25: Young 25-50: Adult 50-100: Old

cdf['Age'] = pd.cut(cdf['Age'], bins = [0, 25, 50, 100], labels = ['Young', 'Adult', 'Old'])

plt.figure(figsize = (12,6))  
sns.countplot(x = 'Age', hue = 'Income', data = cdf , color = 'teal' )

<AxesSubplot:xlabel='Age', ylabel='count'>



**Observations:**

* Majority of the Young people (ages between 0-25) has income less than or equal to 50K.
* Adults has higher percentages of getting an income less than or equal to 50K as compared to Young people and the ratio of Adult people getting an income <=50k and >50k is almonst 1:3 .
* The Old people section has almost (50 - 50)% of people gettting an income less than or equal to 50k and getting an income greater than $50k.

Now lets move on to our next feature i.e Work\_Class . We'll try to visualise how the work\_class is relative to income of an individual.

**Work Class**

cdf['Work\_Class'].value\_counts()

Private 22696  
 Self-emp-not-inc 2541  
 Local-gov 2093  
 ? 1836  
 State-gov 1297  
 Self-emp-inc 1116  
 Federal-gov 960  
 Without-pay 14

Never-worked 7  
Name: Work\_Class, dtype: int64

**Observations**

* There are 22696 individuals who work in Private sector and this is the max of work class in which an individual works.
* A total of (2093+1297+960) = 4350 individuals work under Government , whether Local , State or Federal.
* There is a very less number of people who never worked.
* There are 1836 individuals whose work class is not known.

cdf['Work\_Class'] = cdf['Work\_Class'].astype(object)

We will replace the ('?' Not Known) with Private.

cdf['Work\_Class'].replace(' ?' , ' Private' , inplace = True)

cdf['Work\_Class'].value\_counts()

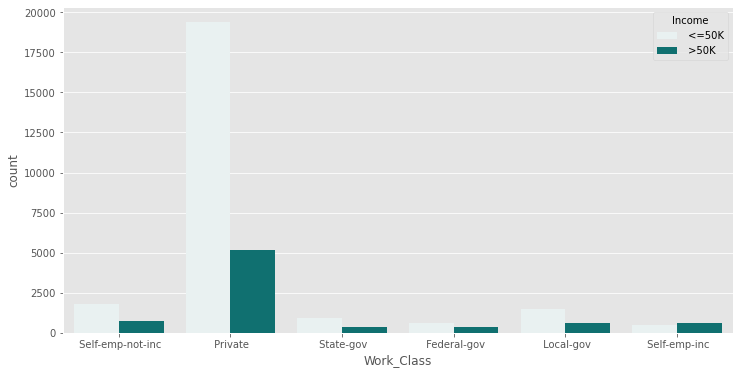
Private 24532  
 Self-emp-not-inc 2541  
 Local-gov 2093  
 State-gov 1297  
 Self-emp-inc 1116  
 Federal-gov 960  
 Without-pay 14  
 Never-worked 7  
Name: Work\_Class, dtype: int64

We'll drop two work classes i.e Without-pay and Never-Worked as their value count is very very less as compared to whole data.

cdf = cdf.drop(cdf[cdf['Work\_Class'] == ' Without-pay'].index)  
cdf = cdf.drop(cdf[cdf['Work\_Class'] == ' Never-worked'].index)

plt.figure(figsize= (12,6))  
sns.countplot(x = 'Work\_Class', hue = 'Income', data = cdf , color = 'teal')

<AxesSubplot:xlabel='Work\_Class', ylabel='count'>



**Observations**

* The private work\_class has the highest sector in which the individuals are having income more than 50k.
* Self-emp-inc is the work\_class in which the individuals having income more than 50k is greater than individuals having income less than 50k.

**Final Weight**

This is the most important feature according to our importance bar . The description of final weight can be found above .

cdf['Final\_Weight'].value\_counts()

123011 13  
164190 13  
203488 13  
121124 12  
148995 12  
 ..  
284211 1  
312881 1  
177711 1  
179758 1  
229376 1  
Name: Final\_Weight, Length: 21634, dtype: int64

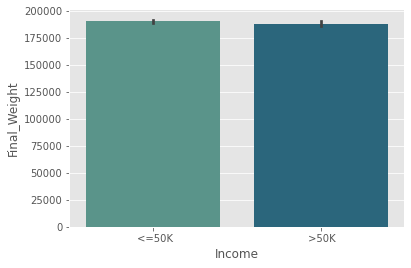
cdf['Final\_Weight'].nunique()

21634

There are 21647 unique values in this feature.

sns.barplot(x = 'Income', y = 'Final\_Weight', data = cdf , palette = 'crest')

<AxesSubplot:xlabel='Income', ylabel='Final\_Weight'>



Final Weight of individuals having income <=50k is a bit high as compared to individuals having income >50k.

**Education and Education Number**

It's a good time to check if there is any relation between Education and Education Number.

education\_classes = cdf['Education'].unique()  
for edu\_class in education\_classes:  
 print("For {}, the Education Number is {}"  
 .format(edu\_class, cdf[cdf['Education'] == edu\_class]['Education\_Number'].unique()))

For Bachelors, the Education Number is [13]  
For HS-grad, the Education Number is [9]  
For 11th, the Education Number is [7]  
For Masters, the Education Number is [14]  
For 9th, the Education Number is [5]  
For Some-college, the Education Number is [10]  
For Assoc-acdm, the Education Number is [12]  
For Assoc-voc, the Education Number is [11]  
For 7th-8th, the Education Number is [4]  
For Doctorate, the Education Number is [16]  
For Prof-school, the Education Number is [15]  
For 5th-6th, the Education Number is [3]  
For 10th, the Education Number is [6]  
For 1st-4th, the Education Number is [2]  
For Preschool, the Education Number is [1]  
For 12th, the Education Number is [8]

From above, We have found that Education Number and Education are just the same. So, we can drop any one column. Also, we'll combine all informations from Preschool to 12th as they can be considered of one class who have no college/university level education.

cdf.drop(['Education\_Number'], axis = 1, inplace = True)  
cdf['Education'].replace([' 11th', ' 9th', ' 7th-8th', ' 5th-6th', ' 10th', ' 1st-4th', ' Preschool', ' 12th'], ' School', inplace = True)  
cdf['Education'].value\_counts()

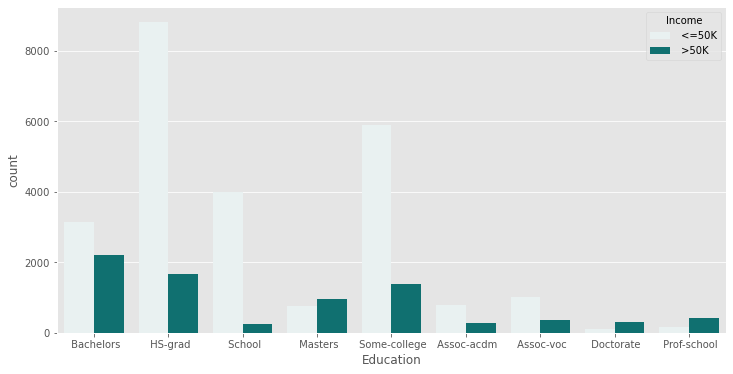
HS-grad 10491  
 Some-college 7286  
 Bachelors 5354  
 School 4248  
 Masters 1723  
 Assoc-voc 1382  
 Assoc-acdm 1066  
 Prof-school 576  
 Doctorate 413  
Name: Education, dtype: int64

**Observations**

* The most number of individuals 10491 are having Hs-grad as their education.
* The lowest number of individuals are having Doctorate as their education level.

plt.figure(figsize = (12,6))  
sns.countplot(x = 'Education', hue = 'Income', data = cdf , color = 'teal')

<AxesSubplot:xlabel='Education', ylabel='count'>



**Observations**

* Bachelors are having the most number of individuals with the highest number of individuals having income greater than 50 k.
* Hs grad is the education level in which there is highest number of individuals having income less than or equal to 50k.
* Masters , Doctorate and Prof-School are the education level in which the number of individuals getting an income of more than 50k is more as compared to individuals having income less than 50k.

**Marital Status**

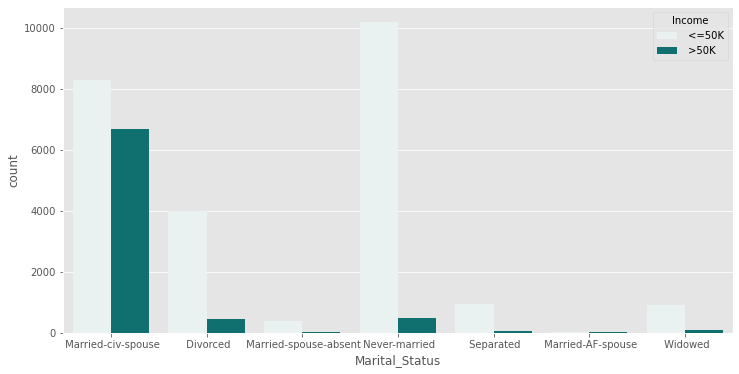
cdf['Marital\_Status'].value\_counts()

Married-civ-spouse 14967  
 Never-married 10673  
 Divorced 4442  
 Separated 1025  
 Widowed 992  
 Married-spouse-absent 417  
 Married-AF-spouse 23  
Name: Marital\_Status, dtype: int64

* Married-civ-spouse :- Married-civ-spouse corresponds to a civilian spouse while *Married-AF-spouse* is a spouse in the Armed Forces.
* Married-spouse-absent :- Married-spouse-absent applies to husbands and wives who answered that they were Now married on the census form but no spouse could be found who could be linked to them in the editing stages.

plt.figure(figsize = (12,6))  
sns.countplot(x = 'Marital\_Status', hue = 'Income', data = cdf , color = 'teal')

<AxesSubplot:xlabel='Marital\_Status', ylabel='count'>



**Observations**

* Married with civilian has the highest number of individuals having an income of greater than 50k.
* Never-married are the individuals having income maximum numbers in having an income less than 50k.

**Relationship**

cdf['Relationship'].value\_counts()

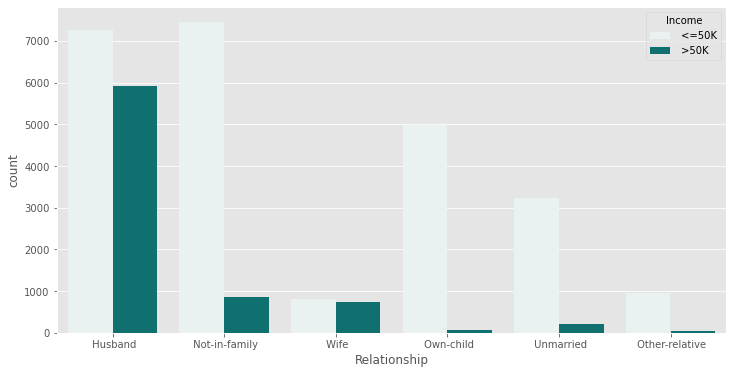
Husband 13189  
 Not-in-family 8303  
 Own-child 5058  
 Unmarried 3444  
 Wife 1564  
 Other-relative 981  
Name: Relationship, dtype: int64

**Observations**

* Individuals living in with husband are the hihest in numbers.
* Then comes to individuals not living with the family.

plt.figure(figsize = (12,6))  
sns.countplot(x = 'Relationship', hue = 'Income', data = cdf , color = 'teal')

<AxesSubplot:xlabel='Relationship', ylabel='count'>



**Observations**

* Individuals living with husband are in the highest in numbers on having an income for both less than or equal to 50k and income of greater than 50k.

**Occupation**

cdf['Occupation'].value\_counts()

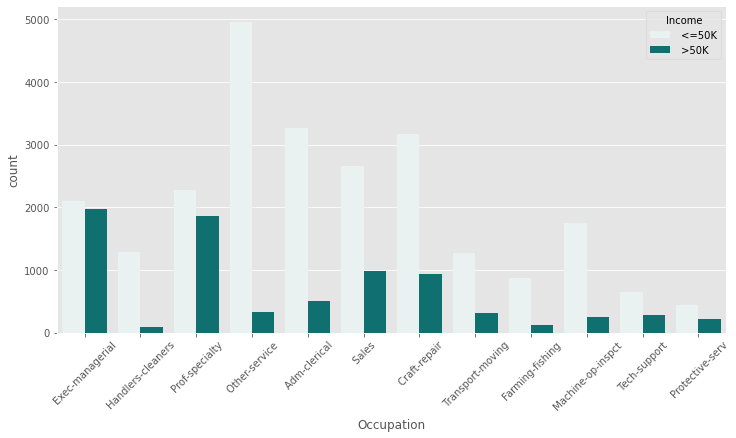
Prof-specialty 4140  
 Craft-repair 4098  
 Exec-managerial 4066  
 Adm-clerical 3766  
 Sales 3650  
 Other-service 3294  
 Machine-op-inspct 2001  
 ? 1836  
 Transport-moving 1596  
 Handlers-cleaners 1369  
 Farming-fishing 988  
 Tech-support 928  
 Protective-serv 649  
 Priv-house-serv 149  
 Armed-Forces 9  
Name: Occupation, dtype: int64

In here we can observe that there are values with '?' and values of Priv-house-serv are very less so we'll conclude all of them to Other-service.

cdf['Occupation'].replace([' Other-service',' ?',' Armed-Forces',' Priv-house-serv'] , ' Other-service' , inplace = True)

plt.figure(figsize=(12,6))  
plt.xticks(rotation = 45)  
sns.countplot(x = 'Occupation', hue = 'Income', data = cdf , color = 'teal')

<AxesSubplot:xlabel='Occupation', ylabel='count'>



cdf.Occupation.value\_counts()

Other-service 5288  
 Prof-specialty 4140  
 Craft-repair 4098  
 Exec-managerial 4066  
 Adm-clerical 3766  
 Sales 3650  
 Machine-op-inspct 2001  
 Transport-moving 1596  
 Handlers-cleaners 1369  
 Farming-fishing 988  
 Tech-support 928  
 Protective-serv 649  
Name: Occupation, dtype: int64

**Observations**

* Exec-mangerial and Prof-speciality are the occupation fields where the incomes greater than 50k are the highest.
* Other-service are the occupation fields where the incomes less than or equal to 50k is maximum.

**Race**

cdf['Race'].value\_counts()

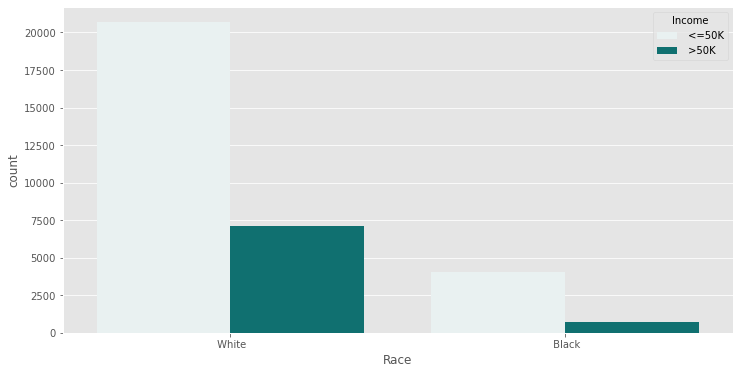
White 27798  
 Black 3121  
 Asian-Pac-Islander 1038  
 Amer-Indian-Eskimo 311  
 Other 271  
Name: Race, dtype: int64

Here the majority of the individuals are White , so we'll group this feature as White and others as Black .

cdf['Race'].replace([' Black', ' Asian-Pac-Islander', ' Amer-Indian-Eskimo', ' Other'],' Black', inplace = True)

plt.figure(figsize = (12,6))  
sns.countplot(x = 'Race', hue = 'Income', data = cdf , color = 'teal')

<AxesSubplot:xlabel='Race', ylabel='count'>



**Observation**

* From above we can observe that from both the datas of White and Black individuals , Whites has more individuals having an income of less than or equal to 50K and greater than 50k.

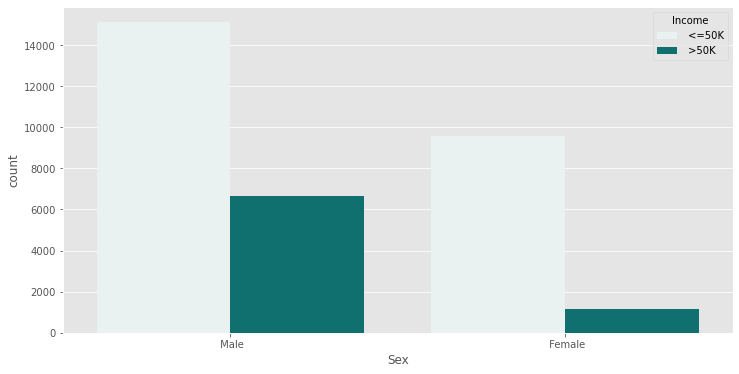
**Sex**

cdf['Sex'].value\_counts()

Male 21775  
 Female 10764  
Name: Sex, dtype: int64

plt.figure(figsize=(12,6))  
sns.countplot(x = 'Sex', hue = 'Income', data = cdf , color = 'teal')

<AxesSubplot:xlabel='Sex', ylabel='count'>



**Observations**

* There are almost twice Male participants as compared to Female participants
* When we compare the two genders and the corresponding income distribution, more percentage of Males individuals have an income of less than or equal to 50k or greater than 50K than Females.

**Capital Gain and Capital Loss**

Rather than having both Capital Gain and Capital Loss feature , we will use their difference as that is more relevant and will be more helpful in our future predictions.

cdf['Capital\_Diff'] = cdf['Capital\_Gain'] - cdf['Capital\_Loss']  
cdf.drop(['Capital\_Gain'], axis = 1, inplace = True)  
cdf.drop(['Capital\_Loss'], axis = 1, inplace = True)

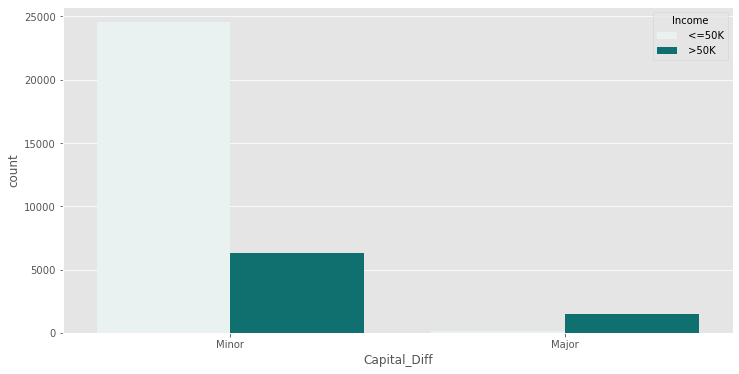
print(cdf['Capital\_Diff'].min())  
print(cdf['Capital\_Diff'].max())

-4356  
99999

The lowest value is -4356 , so we'll take the lowest bin as -5000 and the highest value is 99999 so our highest bin will bw 100000 . Below we are creating two ranges -5000 to 5000 as Minor and 5000 to 10000 as Major.

plt.figure(figsize = (12,6))  
cdf['Capital\_Diff'] = pd.cut(cdf['Capital\_Diff'], bins = [-5000, 5000, 100000], labels = ['Minor', 'Major'])  
sns.countplot(x ='Capital\_Diff' , hue = 'Income', data = cdf , color = 'teal' )

<AxesSubplot:xlabel='Capital\_Diff', ylabel='count'>



**Observations**

* Individuals having minor capital diff are more in numbers on having an income less than or equal to 50k and greater than or equal to 50k.
* Individuals having Major capital diff are more on having an income greater than 50k as compared to individials having an income less than or equal to 50k.

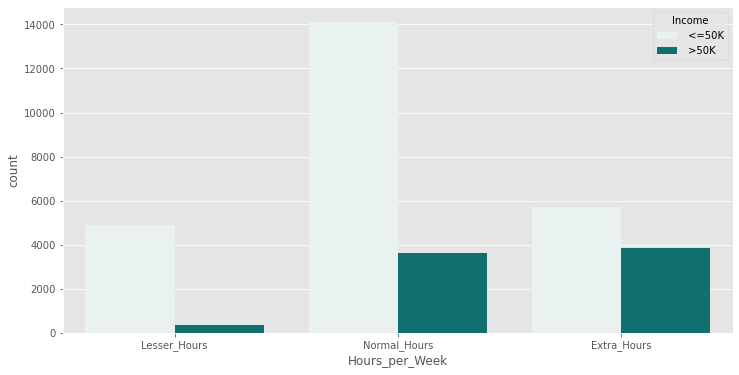
**Hours per Week**

As usually, the working hours are close to 30-40 hours, so we'll create bins of 0-30, 30-40, and 40-100 hrs .

cdf['Hours\_per\_Week'] = pd.cut(cdf['Hours\_per\_Week'], bins = [0, 30, 40, 100], labels = ['Lesser\_Hours', 'Normal\_Hours', 'Extra\_Hours'])

plt.figure(figsize= (12,6))  
sns.countplot(x = 'Hours\_per\_Week', hue = 'Income', data = cdf , color = 'teal')

<AxesSubplot:xlabel='Hours\_per\_Week', ylabel='count'>



**Observations**

* Individuals working normal hours i.e in between 30-40 hrs a week are the most in having an income less than or equal to 50k.
* Individuals working extra hours i.e more than 40 hrs a week more in numbers on having an income greater than 50k as compared to individuals working 30-40 hrs.

**Country**

Countries from which servey's are carried out .

cdf['Country'].value\_counts()

United-States 29149  
 Mexico 643  
 ? 583  
 Philippines 197  
 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  
 Trinadad&Tobago 19  
 Cambodia 19  
 Laos 18  
 Thailand 18  
 Yugoslavia 16  
 Outlying-US(Guam-USVI-etc) 14  
 Hungary 13  
 Honduras 13  
 Scotland 12  
 Holand-Netherlands 1  
Name: Country, dtype: int64

countries = np.array(cdf['Country'].unique())  
countries

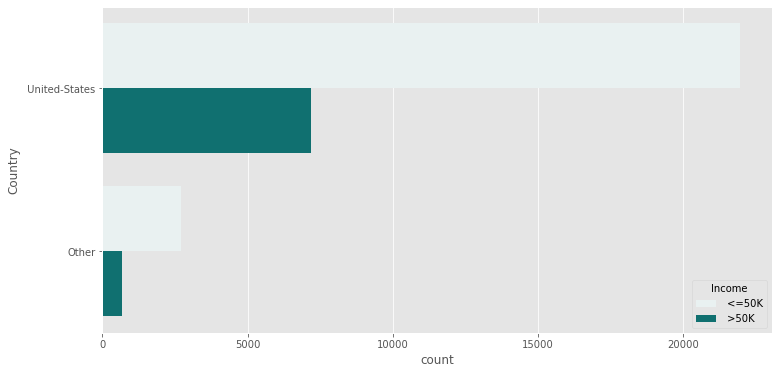
array([' United-States', ' Cuba', ' Jamaica', ' India', ' ?', ' Mexico',  
 ' South', ' Puerto-Rico', ' Honduras', ' England', ' Canada',  
 ' Germany', ' Iran', ' Philippines', ' Italy', ' Poland',  
 ' Columbia', ' Cambodia', ' Thailand', ' Ecuador', ' Laos',  
 ' Taiwan', ' Haiti', ' Portugal', '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

We can observe that out of 32560 , 29149 observations are from United states , so here we'll group all the other countries as other . So basically we will have teo categories United States and Other.

cdf['Country'].replace([' Cuba', ' Jamaica', ' India', ' ?', ' Mexico', ' South', ' Puerto-Rico', ' Honduras', ' England', ' Canada',  
 ' Germany', ' Iran', ' Philippines', ' Italy', ' Poland', ' Columbia', ' Cambodia', ' Thailand', ' Ecuador', ' Laos',  
 ' Taiwan', ' Haiti', ' Portugal', ' Dominican-Republic', ' El-Salvador', ' France', ' Guatemala', ' China', ' Japan',  
 ' Yugoslavia', ' Peru', ' Outlying-US(Guam-USVI-etc)', ' Scotland',' Trinadad&Tobago', ' Greece', ' Nicaragua', ' Vietnam', ' Hong', ' Ireland', ' Hungary', ' Holand-Netherlands'], 'Other' , inplace = True)

plt.figure(figsize = (12,6))  
sns.countplot(y = 'Country', hue = 'Income', data = cdf , color = 'teal')

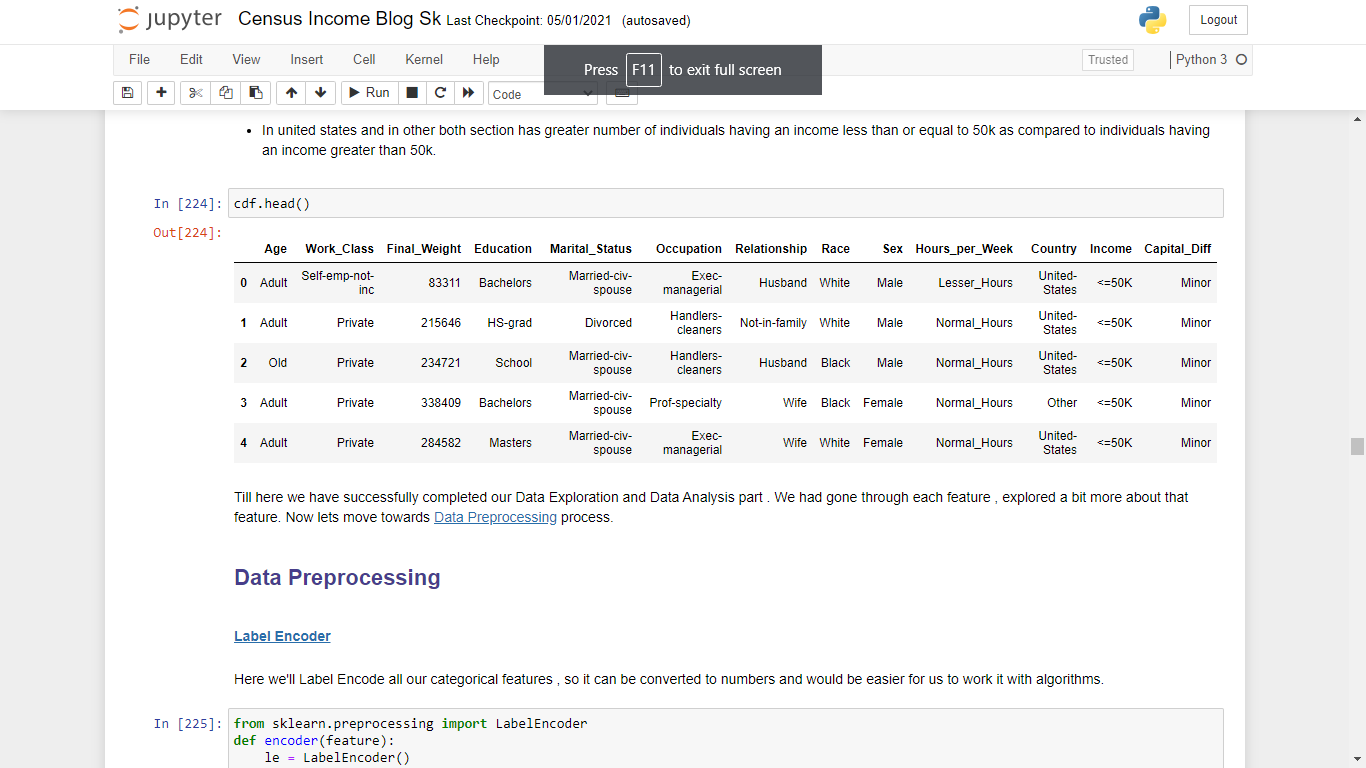
<AxesSubplot:xlabel='count', ylabel='Country'>



**Observations**

* In united states and in other both section has greater number of individuals having an income less than or equal to 50k as compared to individuals having an income greater than 50k.

cdf.head()



Till here we have successfully completed our Data Exploration and Data Analysis part . We had gone through each feature , explored a bit more about that feature. Now lets move towards [Data Preprocessing](https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/) process.

## Data Preprocessing

#### **[Label Encoder](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html)**

Here we'll Label Encode all our categorical features , so it can be converted to numbers and would be easier for us to work it with algorithms.

from sklearn.preprocessing import LabelEncoder  
def encoder(feature):  
 le = LabelEncoder()  
 cdf[feature] = le.fit\_transform(cdf[feature])  
 return cdf[feature].value\_counts()

**Age**

encoder('Age')

0 19686  
1 6452  
2 6401  
Name: Age, dtype: int64

**Work\_Class**

encoder('Work\_Class')

2 24532  
4 2541  
1 2093  
5 1297  
3 1116  
0 960  
Name: Work\_Class, dtype: int64

**Education**

encoder('Education')

4 10491  
8 7286  
2 5354  
7 4248  
5 1723  
1 1382  
0 1066  
6 576  
3 413  
Name: Education, dtype: int64

**Marital\_Status**

encoder('Marital\_Status')

2 14967  
4 10673  
0 4442  
5 1025  
6 992  
3 417  
1 23  
Name: Marital\_Status, dtype: int64

**Occupation**

encoder('Occupation')

6 5288  
7 4140  
1 4098  
2 4066  
0 3766  
9 3650  
5 2001  
11 1596  
4 1369  
3 988  
10 928  
8 649  
Name: Occupation, dtype: int64

**Relationship**

encoder('Relationship')

0 13189  
1 8303  
3 5058  
4 3444  
5 1564  
2 981  
Name: Relationship, dtype: int64

**Race**

encoder('Race')

1 27798  
0 4741  
Name: Race, dtype: int64

**Sex**

encoder('Sex')

1 21775  
0 10764  
Name: Sex, dtype: int64

**Capital\_Difference**

encoder('Capital\_Diff')

1 30891  
0 1648  
Name: Capital\_Diff, dtype: int64

**Hours per Week**

encoder('Hours\_per\_Week')

2 17728  
0 9577

1 5234  
Name: Hours\_per\_Week, dtype: int64

**Country**

encoder('Country')

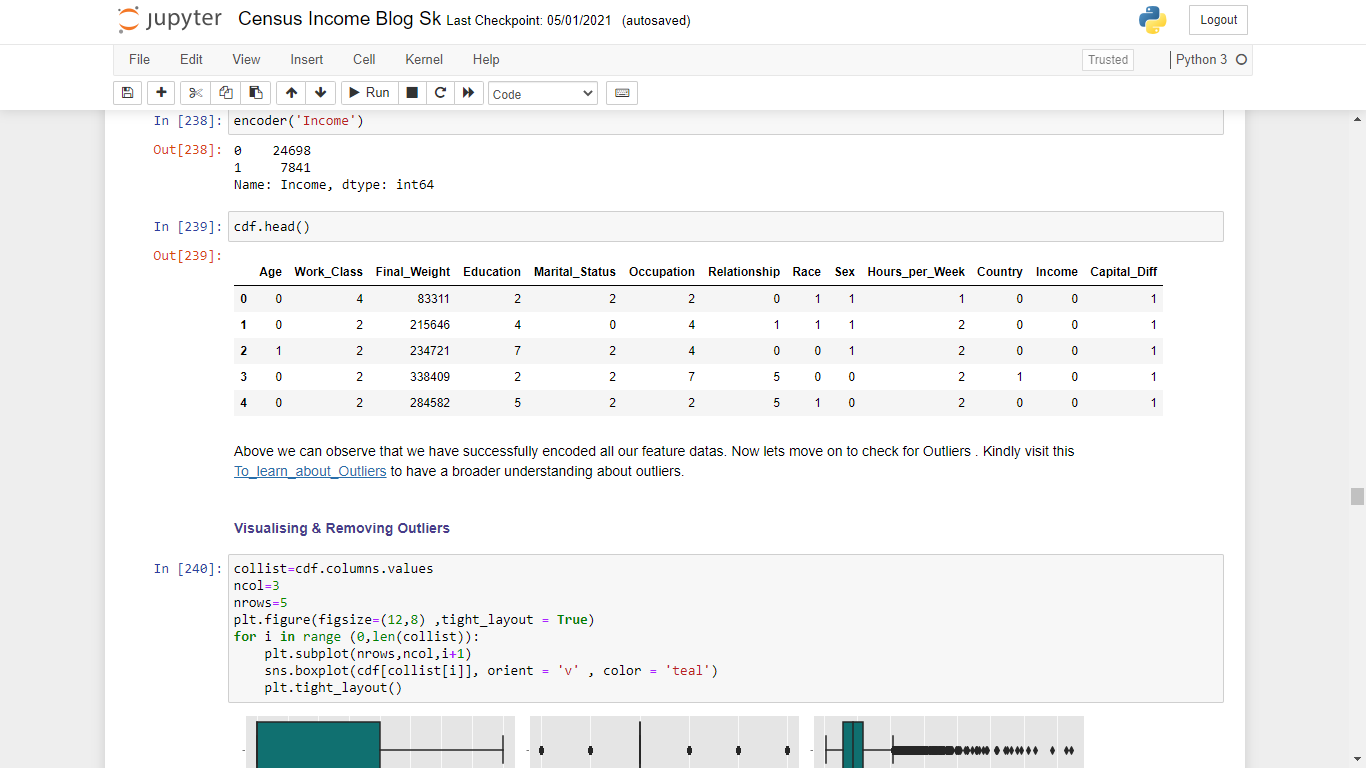
0 29149  
1 3390  
Name: Country, dtype: int64

**Income**

encoder('Income')

0 24698  
1 7841  
Name: Income, dtype: int64

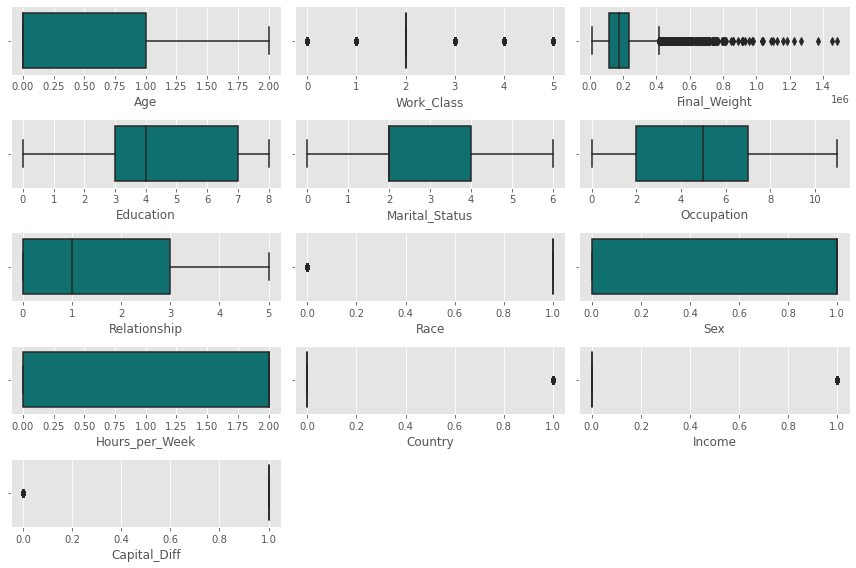
cdf.head()

Age

Above we can observe that we have successfully encoded all our feature datas. Now lets move on to check for Outliers . Kindly visit this [To\_learn\_about\_Outliers](https://datascience.foundation/sciencewhitepaper/knowing-all-about-outliers-in-machine-learning) to have a broader understanding about outliers.

#### **Visualising & Removing Outliers**

collist=cdf.columns.values  
ncol=3  
nrows=5  
plt.figure(figsize=(12,8) ,tight\_layout = True)  
for i in range (0,len(collist)):  
 plt.subplot(nrows,ncol,i+1)  
 sns.boxplot(cdf[collist[i]], orient = 'v' , color = 'teal')  
 plt.tight\_layout()



By observing the boxplot above we can conclude that there are Outliers present in our data , So we will use [Zscore](https://www.statisticshowto.com/probability-and-statistics/z-score/#:~:text=Technically%2C%20a%20z%2Dscore%20is,standard%20deviation%20above%20the%20mean.) to deal with it.

from scipy.stats import zscore  
Z\_score = np.abs(zscore(cdf))

Above we have applied zscore on our dataframe , now we'll create a new dataframe with removed outliers i.e (cdf\_wo).

cdf\_wo = cdf[(Z\_score<3).all(axis = 1)]

print(cdf.shape)  
print(cdf\_wo.shape)

(32539, 13)  
(29347, 13)

As we processed our dataframe and removed outliers from our dataset now lets move on to splitting our values to **Predictor and Target Variable**.

##### **Predictor Variables**

x = cdf\_wo.drop('Income' , axis = 1)  
x

##### 

##### **Target Variable**

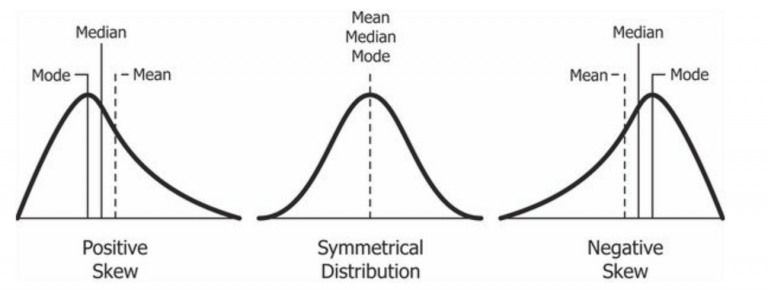
y = cdf\_wo['Income']  
y

0 0  
1 0  
2 0  
3 0  
4 0  
 ..  
32554 0  
32555 0  
32556 1  
32557 0  
32558 0  
Name: Income, Length: 29347, dtype: int32

### Skewness

Skewness is the measure of the asymmetry of an ideally symmetric probability distribution and is given by the third standardized moment. If that sounds way too complex, don’t worry! Let me break it down for you.

In simple words, skewness is the measure of how much the probability distribution of a random variable deviates from the normal distribution. Click to explore more and learn about [Skewness](https://www.analyticsvidhya.com/blog/2020/07/what-is-skewness-statistics/)



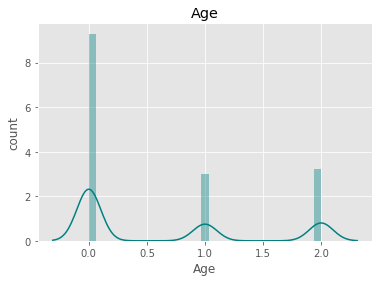
**DataFrame.skew()** - Return unbiased skew over requested axis. [DataFrame.skew()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.skew.html)

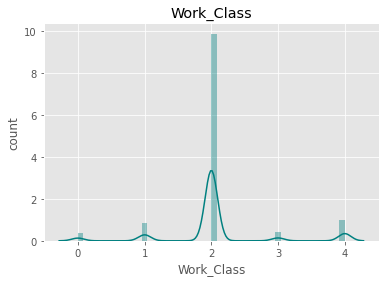
x.skew()

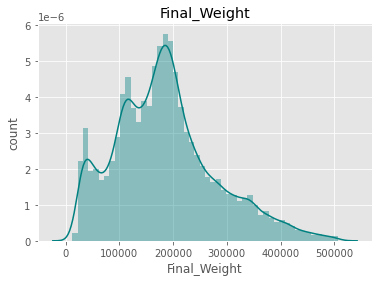
Age 0.827782  
Work\_Class 0.665786  
Final\_Weight 0.628465  
Education -0.069226  
Marital\_Status -0.050466  
Occupation 0.104167  
Relationship 0.743121  
Race -2.015913  
Sex -0.688574  
Hours\_per\_Week -0.528888  
Country 2.555130  
Capital\_Diff 0.000000  
dtype: float64

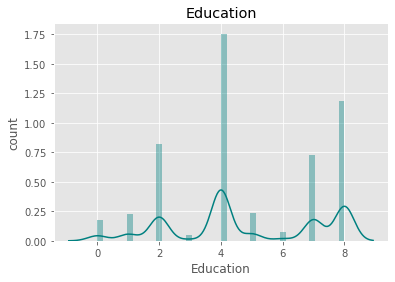
**Distplot** - A distplot plots a univariate distribution of observations. The distplot() function combines the matplotlib hist function with the seaborn kdeplot() and rugplot() functions. Seaborn distplot lets you show a histogram with a line on it. To explore a more about distplot [Distplot](https://seaborn.pydata.org/generated/seaborn.distplot.html).

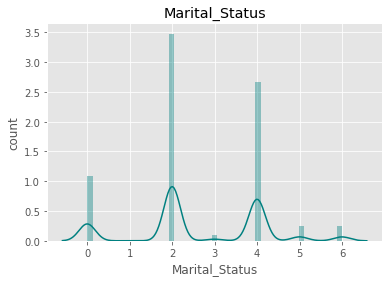
for feature in x :  
 sns.distplot(x[feature] , kde = True , color = 'teal' )  
 plt.xlabel(feature)  
 plt.ylabel("count")  
 plt.title(feature)  
 plt.show()

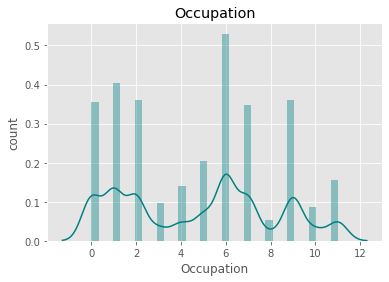


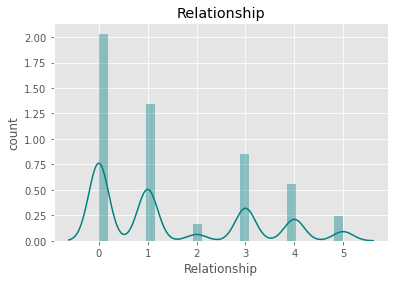


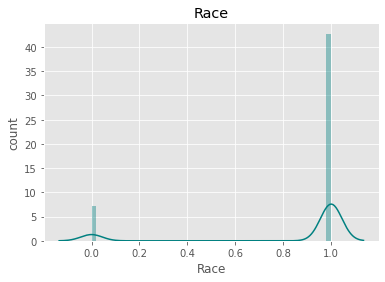


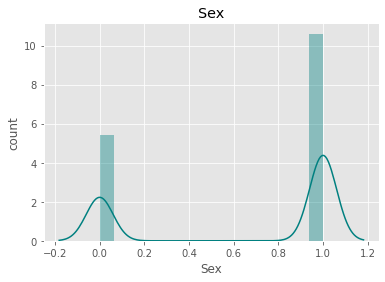


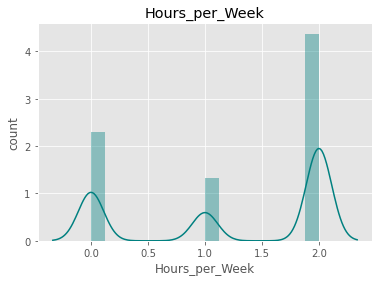


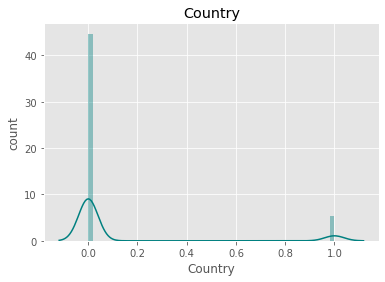


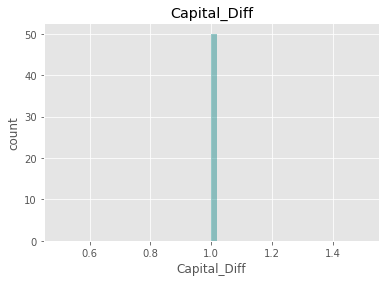












The rule of thumb is: If the skewness is between -0.5 and 0.5, the datas are fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data are moderately skewed. If the skewness is less than -1 or greater than 1, the data are highly skewed.

From above distplot we can conclude that there is skewness in our dataframe features so we need to perform some [Skewness\_removal](https://opendatascience.com/transforming-skewed-data-for-machine-learning/) techniques.

Here we will use [**Power Transformer**](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html) for transforming/removing skewness from our dataset .

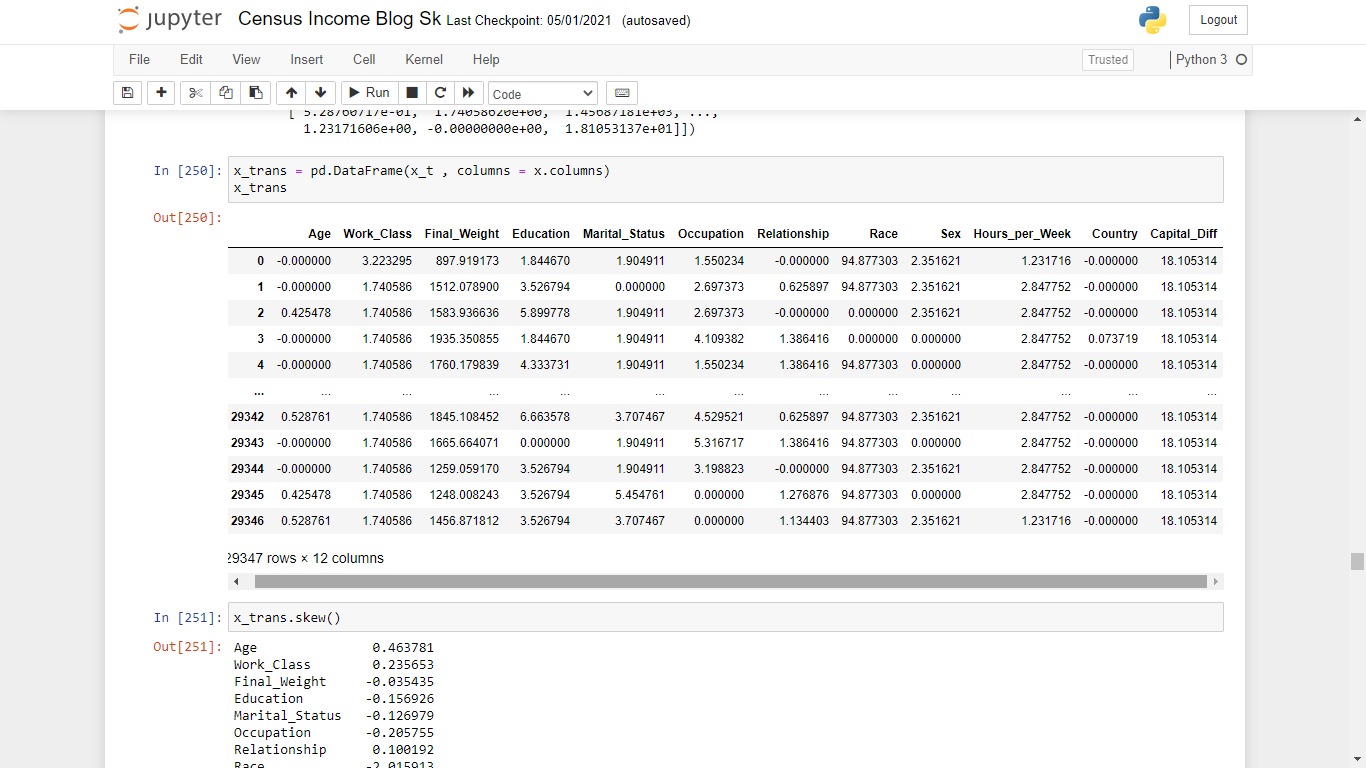
from sklearn.preprocessing import PowerTransformer  
powert = PowerTransformer( method = 'yeo-johnson' , standardize = False)  
x\_t = powert.fit\_transform(x)

The output of power transformer is an array as we can see below , so we'll create a dataframe of this array .

x\_t

array([[-0.00000000e+00, 3.22329537e+00, 8.97919173e+02, ...,  
 1.23171606e+00, -0.00000000e+00, 1.81053137e+01],  
 [-0.00000000e+00, 1.74058620e+00, 1.51207890e+03, ...,  
 2.84775244e+00, -0.00000000e+00, 1.81053137e+01],  
 [ 4.25477738e-01, 1.74058620e+00, 1.58393664e+03, ...,  
 2.84775244e+00, -0.00000000e+00, 1.81053137e+01],  
 ...,  
 [-0.00000000e+00, 1.74058620e+00, 1.25905917e+03, ...,  
 2.84775244e+00, -0.00000000e+00, 1.81053137e+01],  
 [ 4.25477738e-01, 1.74058620e+00, 1.24800824e+03, ...,  
 2.84775244e+00, -0.00000000e+00, 1.81053137e+01],  
 [ 5.28760717e-01, 1.74058620e+00, 1.45687181e+03, ...,  
 1.23171606e+00, -0.00000000e+00, 1.81053137e+01]])

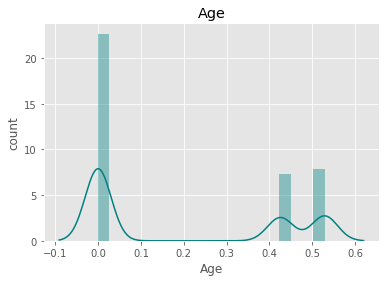
x\_trans = pd.DataFrame(x\_t , columns = x.columns)  
x\_trans



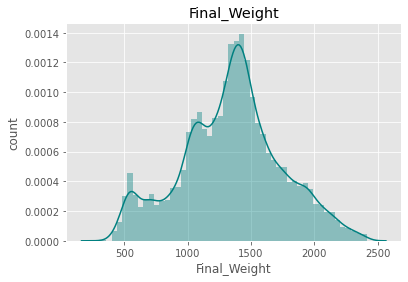
x\_trans.skew()

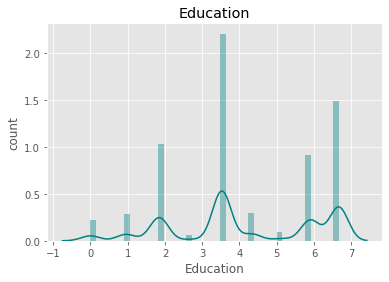
Age 0.463781  
Work\_Class 0.235653  
Final\_Weight -0.035435  
Education -0.156926  
Marital\_Status -0.126979  
Occupation -0.205755  
Relationship 0.100192  
Race -2.015913  
Sex -0.688574  
Hours\_per\_Week -0.448593  
Country 2.555130  
Capital\_Diff 0.000000  
dtype: float64

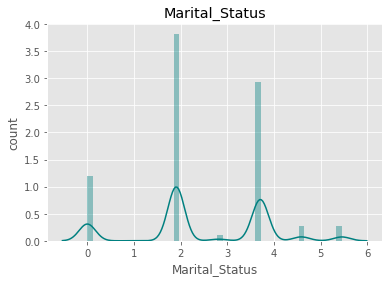
for feature in x\_trans :  
 sns.distplot(x\_trans[feature] , kde = True , color = 'teal' )  
 plt.xlabel(feature)  
 plt.ylabel("count")  
 plt.title(feature)  
 plt.show()

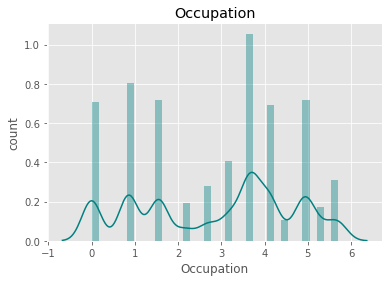


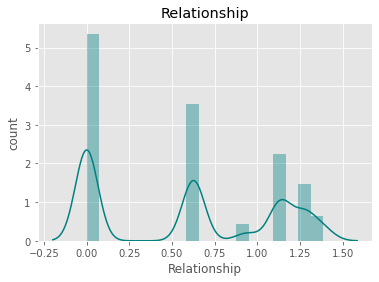


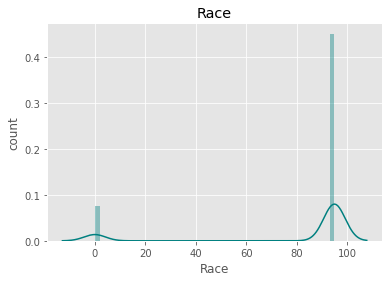


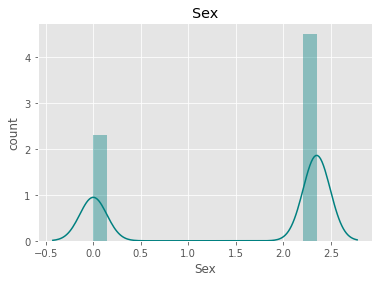


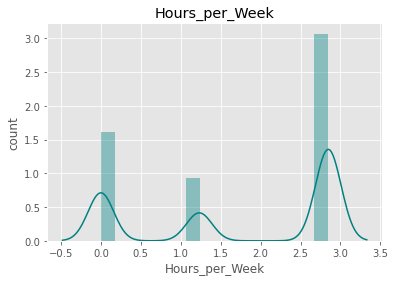


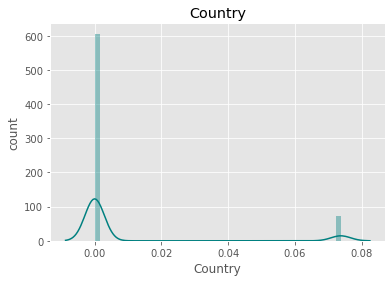


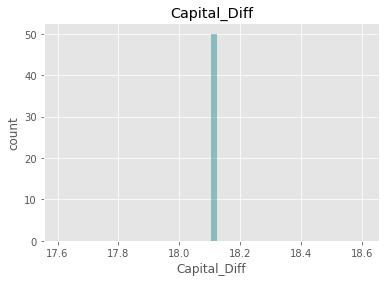












From above we can observe that we have sucessfully removed skewness and outliers from our dataset. Skewness left is of categorical datas. Now lets move on to scale our datas as there are a lot of variation in our dataset .

### Feature Scaling

Feature scaling can vary our results a lot while using certain algorithms and have a minimal or no effect in others. To understand this, let’s look why features need to be scaled, varieties of scaling methods and when we should scale our features.

*What is Feature Scaling?*

It refers to putting the values in the same range or same scale so that no variable is dominated by the other.

*Why Scaling?* Most of the times, our dataset will contain features highly varying in magnitudes, units and range. But since, most of the machine learning algorithms use Euclidean distance between two data points in their computations, this is a problem. If left alone, these algorithms only take in the magnitude of features neglecting the units. The results would vary greatly between different units, 5kg and 5000gms. The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes. To suppress this effect, we need to bring all features to the same level of magnitudes. This can be achieved by scaling.

So here we'll use Standard scaling to scale our dataset. kinldy go through this [standard scaling](https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/#:~:text=Standardization%20is%20another%20scaling%20technique,has%20a%20unit%20standard%20deviation.) article to have a broader understanding of scaling and normalisation.

Gaussian's distribution with zero mean and unit variance is standard scaling. Statistical formula of standardisation is as follows:-

Stand_eq.gif

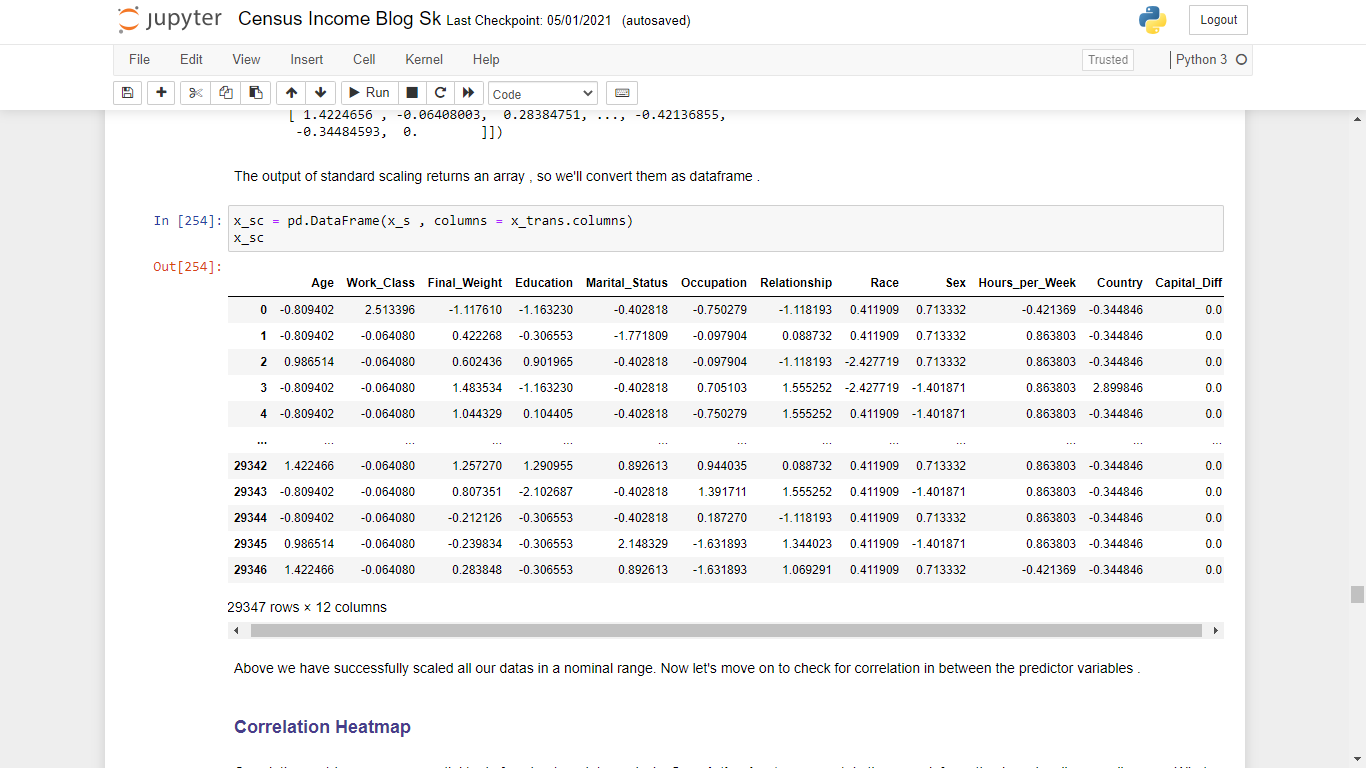
[**Standard scaling sklearn**](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)

scaler = StandardScaler()  
x\_s = scaler.fit\_transform(x\_trans)  
x\_s

array([[-0.80940221, 2.51339555, -1.11760963, ..., -0.42136855,  
 -0.34484593, 0. ],  
 [-0.80940221, -0.06408003, 0.42226777, ..., 0.86380259,  
 -0.34484593, 0. ],  
 [ 0.98651424, -0.06408003, 0.60243605, ..., 0.86380259,  
 -0.34484593, 0. ],  
 ...,  
 [-0.80940221, -0.06408003, -0.21212642, ..., 0.86380259,  
 -0.34484593, 0. ],  
 [ 0.98651424, -0.06408003, -0.23983432, ..., 0.86380259,  
 -0.34484593, 0. ],  
 [ 1.4224656 , -0.06408003, 0.28384751, ..., -0.42136855,  
 -0.34484593, 0. ]])

The output of standard scaling returns an array , so we'll convert them as dataframe .

x\_sc = pd.DataFrame(x\_s , columns = x\_trans.columns)  
x\_sc



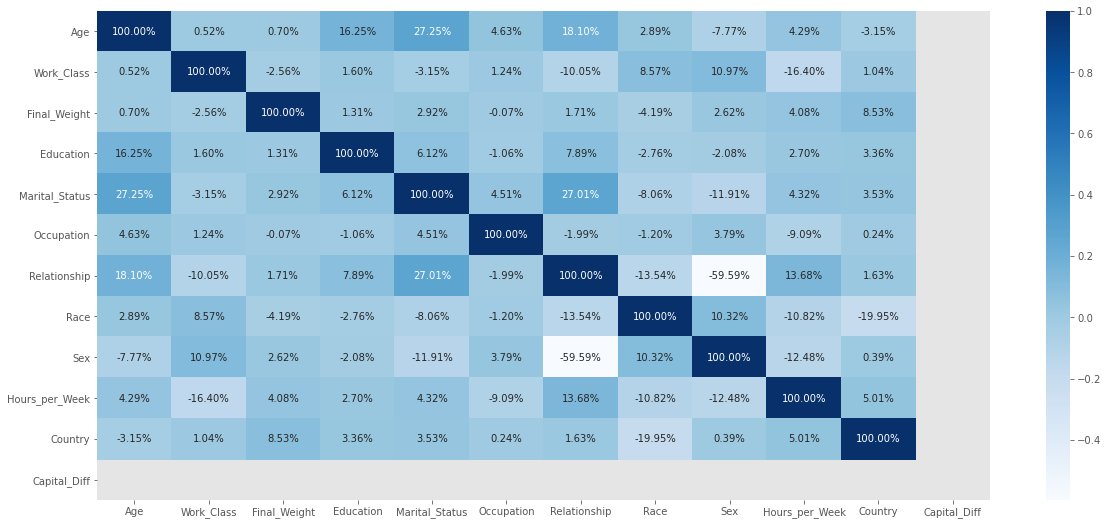
Above we have successfully scaled all our datas in a nominal range. Now let's move on to check for correlation in between the predictor variables .

### Correlation Heatmap

Correlation matrices are an essential tool of exploratory data analysis. **Correlation heatmaps** contain the same information in a visually appealing way. What more: they show in a glance which variables are correlated, to what degree, in which direction, and alerts us to potential multicollinearity problems. Have a look on this article on [heatmaps](https://medium.com/@szabo.bibor/how-to-create-a-seaborn-correlation-heatmap-in-python-834c0686b88e#:~:text=From%20now%20on%2C%20we%20are,in%20a%20visually%20appealing%20way.) to have a broader understanding of heatmaps.

plt.figure(figsize= (20,9))  
sns.heatmap(x\_sc.corr() , cmap = 'Blues' , annot = True , fmt = '.2%')

<AxesSubplot:>



From above we can observe that our predictor variables are not very much correlated with each other , so we will not have a need of [Principal Component Analysis (PCA)](https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html).

**Importance Bar Graph**

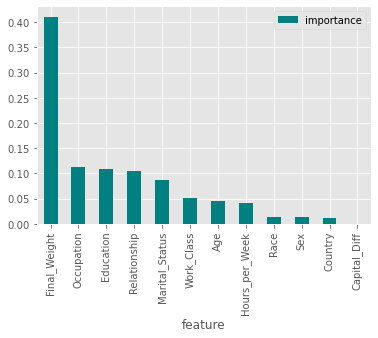
Tree based machine learning algorithms such as Random Forest and XGBoost come with a feature importance attribute that outputs an array containing a value between 0 and 1 for each feature representing how useful the model found each feature in trying to predict the target. This graph shows the importance of each feature in predicting target variable .

X\_train,X\_test,y\_train,y\_test=train\_test\_split(x\_sc,y,test\_size=.30,random\_state=42)  
rf=RandomForestClassifier()  
rf.fit(X\_train,y\_train)  
importances = pd.DataFrame({'feature':x\_sc.columns,'importance':np.round(rf.feature\_importances\_,3)})  
importances = importances.sort\_values('importance',ascending=False).set\_index('feature')  
importances.head(15)

importance  
feature   
Final\_Weight 0.411  
Occupation 0.112  
Education 0.109  
Relationship 0.105  
Marital\_Status 0.086  
Work\_Class 0.051  
Age 0.046  
Hours\_per\_Week 0.042  
Race 0.013  
Sex 0.013  
Country 0.012  
Capital\_Diff 0.000

importances.plot.bar(color = 'teal')

<AxesSubplot:xlabel='feature'>



**Observations**

* From above importance graph we can conclude that final weight is highly important in predicting our target.
* Capital\_diff is very very less important in predicting our target variable i.e income.

## Machine Learning Models

In here we will use various classification algorithm to predict our target. Let's have an overview of the algorithms we will use for our predictions. To read more about these algorithms , just click on the algorithms name.

* [Logistic Regression](https://machinelearningmastery.com/logistic-regression-for-machine-learning/) :- Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a *binary value* (0 or 1) rather than a numeric value. Below is an example logistic regression equation:

*y = e^(b0 + b1*x) / (1 + e^(b0 + b1*x))*

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x).

* [K-Nearest Neighbors](https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn) :- In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm. This is the simplest case. Suppose P1 is the point, for which label needs to predict. First, knn find the one closest point to P1 and then the label of the nearest point assigned to P1. Suppose P1 is the point, for which label needs to predict. First, knn find the k closest point to P1 and then classify points by majority vote of its k neighbors. Each object votes for their class and the class with the most votes is taken as the prediction. For finding closest similar points, knkn find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance.
* [Support Vector Classifier (SVC)](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/) :- In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well. Support Vectors are simply the co-ordinates of individual observation. The SVM classifier is a frontier which best segregates the two classes (hyper-plane/ line).
* [Random Forest Classifier](https://www.datacamp.com/community/tutorials/random-forests-classifier-python) :- It technically is an ensemble method (based on the divide-and-conquer approach) of decision trees generated on a randomly split dataset. This collection of decision tree classifiers is also known as the forest. The individual decision trees are generated using an attribute selection indicator such as information gain, gain ratio, and Gini index for each attribute. Each tree depends on an independent random sample. In a classification problem, each tree votes and the most popular class is chosen as the final result.
* [AdaBoost Classifier](https://www.datacamp.com/community/tutorials/adaboost-classifier-python) :- Ada-boost or Adaptive Boosting is one of ensemble boosting classifier proposed by Yoav Freund and *Robert Schapire* in 1996. It combines multiple classifiers to increase the accuracy of classifiers. AdaBoost is an iterative ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier. The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations.
* [Decision Tree Classifier](https://www.datacamp.com/community/tutorials/decision-tree-classification-python) :- A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making.
* [Gaussian](https://www.geeksforgeeks.org/naive-bayes-classifiers/) :- Naive Bayes classifiers are a collection of classification algorithms based on Bayes’ Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

### Finding Best Random State

An algorithm might have multiple points that introduce randomness to the process and thus introduce randomness to the result. One method to make sure our results are constant is to set every possible *random\_state* available in the functions that we use.

### Train\_Test\_Split

The [Train\_Test\_Split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

In finding best random state we are using random forest classifier as our classification algorithm.

maxAccu=0  
maxRS=0  
for i in range(0,200):  
 X\_train,X\_test,y\_train,y\_test=train\_test\_split(x\_sc,y,test\_size=.30,random\_state=i)  
 rf=RandomForestClassifier()  
 rf.fit(X\_train,y\_train)  
 predrf=rf.predict(X\_test)  
 acc=accuracy\_score(y\_test,predrf)  
 if acc>maxAccu:  
 maxAccu=acc  
 maxRS=i  
   
print('Best accuracy is ',maxAccu, 'on random state ',maxRS)

Best accuracy is 0.8291879613855764 on random state 75

Below we'll apply different classifier algorithms to find the best model for our further predictions.

X\_train,X\_test,y\_train,y\_test=train\_test\_split(x\_sc,y,test\_size=.30,random\_state=75)  
  
models = [LogisticRegression(),  
 KNeighborsClassifier(),  
 SVC(),  
 RandomForestClassifier(),  
 AdaBoostClassifier(),  
 DecisionTreeClassifier(),  
 GaussianNB()  
 ]  
  
names = ['LogisticRegression','K Nearest Neighbor','Support Vector Classifier','Random Forest','AdaBoost Classifier',  
 'Decision Tree Classifier' , 'GaussianNB' ]  
  
for model,name in zip(models,names):  
 fit = model.fit(X\_train , y\_train)  
 y\_predicted = model.predict(X\_test)

score = model.score(X\_train , y\_train)

print(name ," - " ,score)  
 print("Accuracy:",accuracy\_score(y\_predicted, y\_test))  
 print("Confusion Matrix:\n",confusion\_matrix(y\_predicted, y\_test))  
 print("\t\tclassification report")  
 print("-" \* 52)  
 print(classification\_report(y\_predicted , y\_test))

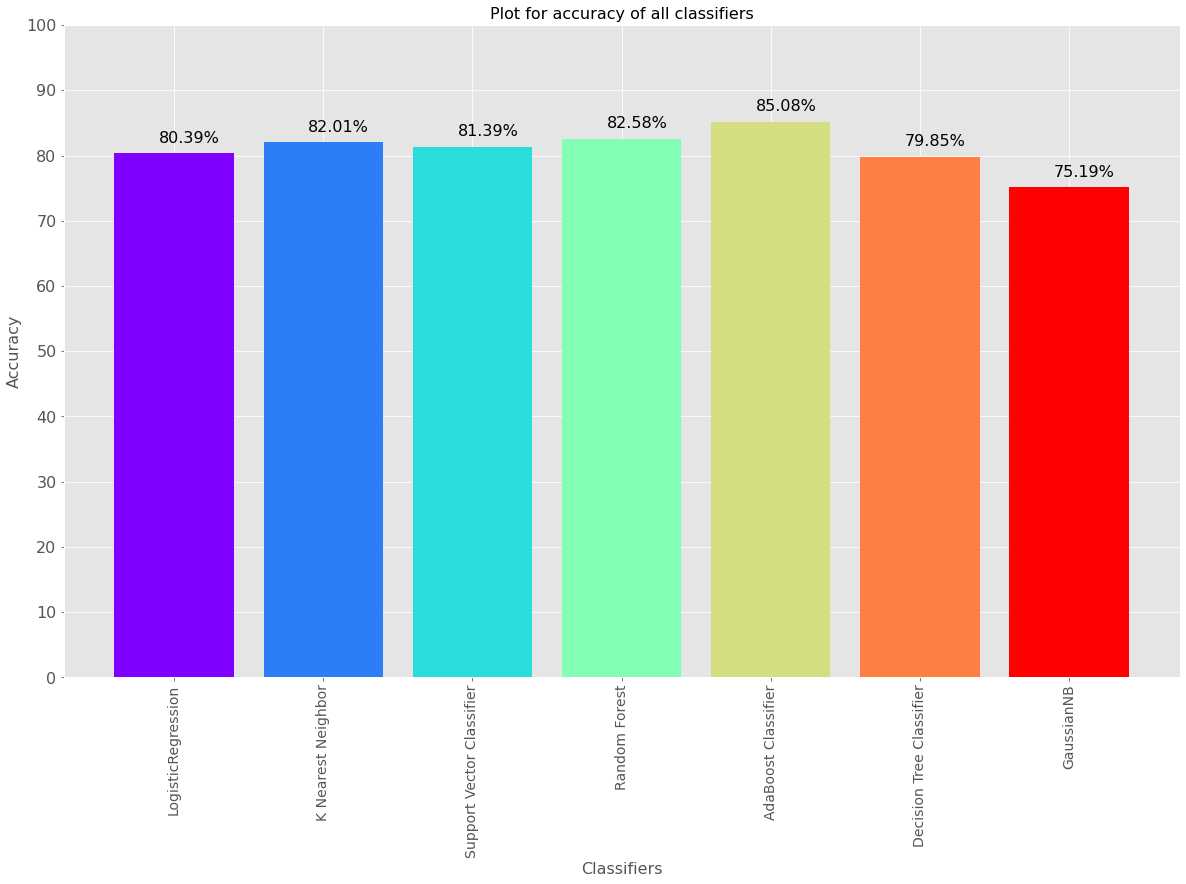
**LogisticRegression**  - 0.803086359653393  
Accuracy: 0.8039750141964793  
Confusion Matrix:  
 [[6786 1497]  
 [ 229 293]]  
 classification report  
----------------------------------------------------  
 precision recall f1-score support  
  
 0 0.97 0.82 0.89 8283  
 1 0.16 0.56 0.25 522  
  
 accuracy 0.80 8805  
 macro avg 0.57 0.69 0.57 8805  
weighted avg 0.92 0.80 0.85 8805  
  
**K Nearest Neighbor** - 0.8679778015772563  
Accuracy: 0.8201022146507666  
Confusion Matrix:  
 [[6377 946]  
 [ 638 844]]  
 classification report  
----------------------------------------------------  
 precision recall f1-score support  
  
 0 0.91 0.87 0.89 7323  
 1 0.47 0.57 0.52 1482  
  
 accuracy 0.82 8805  
 macro avg 0.69 0.72 0.70 8805  
weighted avg 0.84 0.82 0.83 8805  
  
**Support Vector Classifier** - 0.8142342517768474  
Accuracy: 0.81396933560477  
Confusion Matrix:  
 [[6741 1364]  
 [ 274 426]]  
 classification report  
----------------------------------------------------  
 precision recall f1-score support  
  
 0 0.96 0.83 0.89 8105  
 1 0.24 0.61 0.34 700  
  
 accuracy 0.81 8805  
 macro avg 0.60 0.72 0.62 8805  
weighted avg 0.90 0.81 0.85 8805

**Random Forest** - 0.9995618732353228  
Accuracy: 0.8277115275411698  
Confusion Matrix:  
 [[6389 891]  
 [ 626 899]]  
 classification report  
----------------------------------------------------  
 precision recall f1-score support  
  
 0 0.91 0.88 0.89 7280  
 1 0.50 0.59 0.54 1525  
  
 accuracy 0.83 8805  
 macro avg 0.71 0.73 0.72 8805  
weighted avg 0.84 0.83 0.83 8805  
  
**AdaBoost Classifier** - 0.8450491675591472  
Accuracy: 0.8508801817149347  
Confusion Matrix:  
 [[6581 879]  
 [ 434 911]]  
 classification report  
----------------------------------------------------  
 precision recall f1-score support  
  
 0 0.94 0.88 0.91 7460  
 1 0.51 0.68 0.58 1345  
  
 accuracy 0.85 8805  
 macro avg 0.72 0.78 0.75 8805  
weighted avg 0.87 0.85 0.86 8805  
  
**Decision Tree Classifier** - 0.9995618732353228  
Accuracy: 0.7964792731402612  
Confusion Matrix:  
 [[6077 854]  
 [ 938 936]]  
 classification report  
----------------------------------------------------  
 precision recall f1-score support  
  
 0 0.87 0.88 0.87 6931  
 1 0.52 0.50 0.51 1874  
  
 accuracy 0.80 8805  
 macro avg 0.69 0.69 0.69 8805  
weighted avg 0.79 0.80 0.79 8805  
  
**GaussianNB**  - 0.7495862136111382  
Accuracy: 0.7519591141396934  
Confusion Matrix:  
 [[5401 570]  
 [1614 1220]]  
 classification report  
----------------------------------------------------  
 precision recall f1-score support  
  
 0 0.77 0.90 0.83 5971  
 1 0.68 0.43 0.53 2834  
  
 accuracy 0.75 8805  
 macro avg 0.73 0.67 0.68 8805  
weighted avg 0.74 0.75 0.73 8805

**Accuracy Plot**

Plotting Accuracies of different Algorithms to find the perfect one for our model.

accuracies = [80.39 , 82.01 , 81.39 , 82.58 , 85.08 , 79.85 , 75.19]  
  
plt.figure(figsize = (20,12))  
colors = rainbow(np.linspace(0, 1, len(models)))  
barplot = plt.bar(x = names , height = accuracies , color = colors ,tick\_label = names)  
plt.yticks([0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100], fontsize = 16)  
plt.xticks( rotation = 90 , fontsize = 14)  
plt.xlabel("Classifiers", fontsize = 16)  
plt.ylabel("Accuracy", fontsize = 16)  
plt.title("Plot for accuracy of all classifiers", fontsize = 16)  
for i, bar in enumerate(barplot):  
 plt.text(bar.get\_x() + bar.get\_width()/2 - 0.1,   
 bar.get\_height()\*1.02,   
 s = '{:.2f}%'.format(accuracies[i]),   
 fontsize = 16)



After carefully observing the plot above we can conclude that AdaBoostClassifier has the best accuracy and confusion\_matrix scores. So we'll use **AdaBoost Classifier** for our future predictions.

#### **Training AdaBoostClassifier again**

X\_train,X\_test,y\_train,y\_test=train\_test\_split(x\_sc,y,test\_size=.30,random\_state=13)  
  
adaboost\_classifier = AdaBoostClassifier( n\_estimators= 1000 )  
adaboost\_classifier.fit(X\_train , y\_train)  
y\_prediction = adaboost\_classifier.predict(X\_test)  
adaboost\_classifier.score(X\_train, y\_train)  
acc\_adaboost\_classifier = round(adaboost\_classifier.score(X\_train, y\_train) \* 100, 2)  
print(round(acc\_adaboost\_classifier,2,), "%")

84.9 %

#### **Hyperparameter tuning** of AdaBoost Classifier.

Parameters which define the model architecture are referred to as hyperparameters and thus this process of searching for the ideal model architecture is referred to as hyperparameter tuning. There are two types of hyperparameter tuning - Grid Search CV and Randomised Search , here we'll use grid search cv for our further tuning. Read more about [Hyperparameter Tuning](https://www.geeksforgeeks.org/hyperparameter-tuning/).

param\_grid = {'learning\_rate':[0.001, 0.10, 0.1, 1] , 'n\_estimators':range(50, 200, 50)}  
  
  
ab = AdaBoostClassifier( random\_state = 19)  
  
  
grid\_ab = GridSearchCV(ab , param\_grid, scoring = 'accuracy')  
grid\_ab.fit(X\_train, y\_train)  
  
print("Best Hyper Parameters:\n",grid\_ab.best\_params\_)  
print("training accuracy:\n",grid\_ab.best\_score\_)  
ab\_grid\_pred = grid\_ab.best\_estimator\_.predict(X\_test)  
  
print("Accuracy:",accuracy\_score(ab\_grid\_pred , y\_test))  
  
print("Confusion Matrix:\n",confusion\_matrix(ab\_grid\_pred , y\_test))  
print("\t\tclassification report")  
print("-" \* 52)  
print(classification\_report(ab\_grid\_pred , y\_test))

Best Hyper Parameters:  
 {'learning\_rate': 1, 'n\_estimators': 150}  
training accuracy:  
 0.8447082816047515  
Accuracy: 0.8500851788756388  
Confusion Matrix:  
 [[6584 889]  
 [ 431 901]]  
 classification report  
----------------------------------------------------  
 precision recall f1-score support  
  
 0 0.94 0.88 0.91 7473  
 1 0.50 0.68 0.58 1332  
  
 accuracy 0.85 8805  
 macro avg 0.72 0.78 0.74 8805  
weighted avg 0.87 0.85 0.86 8805

**Confusion Matrix**

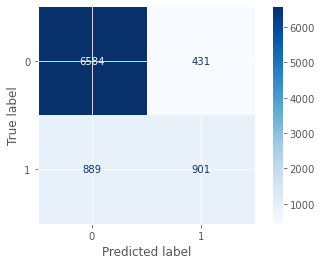
This is used for binary classification as True Negative , False Negaitve , True Positive and False Positive . Read more about [confusion matrix](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html).

predictions = cross\_val\_predict(grid\_ab, X\_train, y\_train, cv=3)  
confusion\_matrix(y\_train, predictions)

array([[15297, 1040],  
 [ 2162, 2043]], dtype=int64)

plot\_confusion\_matrix( grid\_ab ,X\_test , y\_test , cmap = 'Blues' )

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x18c5c6a9610>

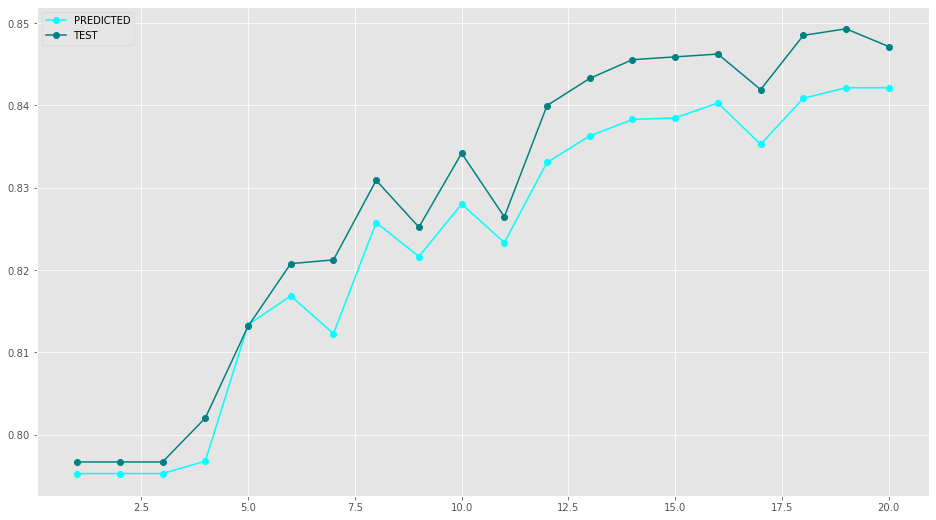


#### **PLotting y\_test and y\_predicted for our datset .**

After successfully applying classifier algortihms on our datasets , Let's plot our actual y\_test and predicted y, and have a look on the difference in values of both .

train\_scores, test\_scores = list(), list()  
values = [i for i in range(1, 21)]  
  
for i in values:  
  
 model = AdaBoostClassifier(n\_estimators=i)  
   
 model.fit(X\_train, y\_train)  
   
 train\_y\_pred = model.predict(X\_train)  
 train\_accuracy = accuracy\_score(y\_train,train\_y\_pred)  
 train\_scores.append(train\_accuracy)  
   
 test\_y\_pred = model.predict(X\_test)  
 test\_accuracy = accuracy\_score(y\_test, test\_y\_pred)  
 test\_scores.append(test\_accuracy)  
   
   
   
 print('>%d, train: %.3f, test: %.3f' % (i, train\_accuracy, test\_accuracy))  
  
plt.figure(figsize=(16,9))  
plt.plot(values, train\_scores, '-o', label='PREDICTED' , color = 'aqua')  
plt.plot(values, test\_scores, '-o', label='TEST' , color = 'teal')  
plt.legend()  
plt.show()

>1, train: 0.795, test: 0.797  
>2, train: 0.795, test: 0.797  
>3, train: 0.795, test: 0.797  
>4, train: 0.797, test: 0.802  
>5, train: 0.813, test: 0.813  
>6, train: 0.817, test: 0.821  
>7, train: 0.812, test: 0.821  
>8, train: 0.826, test: 0.831  
>9, train: 0.822, test: 0.825  
>10, train: 0.828, test: 0.834  
>11, train: 0.823, test: 0.826  
>12, train: 0.833, test: 0.840  
>13, train: 0.836, test: 0.843  
>14, train: 0.838, test: 0.846  
>15, train: 0.838, test: 0.846  
>16, train: 0.840, test: 0.846  
>17, train: 0.835, test: 0.842  
>18, train: 0.841, test: 0.848  
>19, train: 0.842, test: 0.849  
>20, train: 0.842, test: 0.847

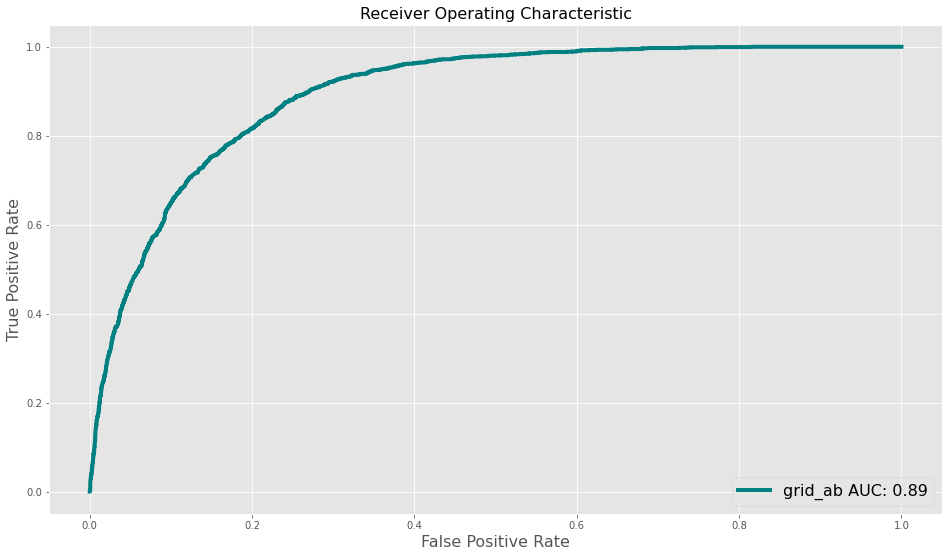


#### **Receiver Operator Characteristic\_Area Under Curve (ROC\_AUC) Curve**

There is a lot more to learn about Roc\_Auc curve , visit the link attached below to eplore about roc\_auc curve . For now the ROC\_AUC curve helps us visualize how well our machine learning classifier is performing. Go through this article to have in depth knowledge of [Roc\_Auc curve](https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/).

probs = grid\_ab.predict\_proba(X\_test)  
  
probs = probs[:, 1]  
fpr, tpr, thresholds = roc\_curve(y\_test, probs)  
roc\_auc = auc(fpr, tpr)  
label = 'grid\_ab' + ' AUC:' + ' {0:.2f}'.format(roc\_auc)  
plt.figure(figsize=(16,9))   
plt.plot(fpr, tpr,label = label , linewidth = 4 ,color = 'teal')  
   
plt.xlabel('False Positive Rate', fontsize = 16)  
plt.ylabel('True Positive Rate', fontsize = 16)  
plt.title('Receiver Operating Characteristic', fontsize = 16)  
plt.legend(loc = 'lower right', fontsize = 16)

<matplotlib.legend.Legend at 0x18c5bc46fa0>



**Saving the Machine Learning Model.**

There are various ways through which we can save a machine learning model , in here we are using joblib to save our best model .Read more about saving a machine learning model [here](https://www.geeksforgeeks.org/saving-a-machine-learning-model/)

import joblib  
joblib.dump(grid\_ab , 'AdaBoostClassifier.pkl')

['AdaBoostClassifier.pkl']

Thank You