Project Name: Census Income Predictions

Author: Aniruddha Sawant

- **Problem Definition.**

Computing or foreseeing an individual's pay is vital. The pay of individual concludes the development and success of a country and it very well may be helpful in different cases to be specific promoting, research, etc. In this undertaking we will anticipate the individual's pay in view of different highlights and factors like his Age, Education, Occupation, Sex, etc.

To anticipate this information, we want to make an AI model for which we required information and we have the wellspring of such information. This information was separated from the 1994 Census agency data set by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics). A bunch of sensibly clean records was separated utilizing the accompanying circumstances: ((AGE>16) and (AGI>100) and (AFNLWGT>1) and (HRSWK>0)).

The expectation task is to decide if an individual makes more than $50K a year in view of given factors.

Below is the snapshot of a dataset: -

In [3]:
```
df = pd.read_csv('https://raw.githubusercontent.com/dsrscientist/dataset1/master/census_income.csv')
df
```

Out[3]:

Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship	Race	Sex	Capital_gain	Capital_loss	Hours_per_week	Native_country	Income
83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K

In this Dataset Income is the Label. We are building a model to Forecast the Income. There are complete 14 features which are: -

1. Age
2. Workclass
3. Fnlwgt
4. Education
5. Education_num
6. Marital_status
7. Occupation
8. Relationship
9. Race
10. Sex
11. Capital_gain
12. Capital_loss
13. Hours_per_week
14. Native_country

- **Data Analysis.**

- We have total 15 columns including the label i.e Income column.
  1. Age - Age of an individual
  2. Workclass - Work class of an individual whether he is a private employee or independently employed regardless of pay.
  3. Fnlwgt - 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 estimates of the population 16+ for each state.
     - Controls for Hispanic Origin by age and sex.
     - Controls by Race, age and sex.
  4. Education - It addresses the training of an individual concerning his certification.
  5. Education_num - This section shows the number of trainings.
  6. Marital_status - This section shows the marital status of an individual.
  7. Occupation - Occupation of an individual.
  8. Relationship - This section shows the relationship of an individual in the family.
  9. Race - Race of an individual.
  10. Sex - Gender of an individual.
  11. Capital_gain - This section shows how much benefit has been made by an individual.
  12. Capital_loss - This section shows how much misfortune has been made by an individual.
  13. Hours_per_week - Daily long periods of working.
  14. Native_country - Nationality of an individual.
  15. Income - Income of an individual. Name in the dataset.

  Each Features have some influence in predicting the income of a person. It is necessary to know which features have greater impact on income and which does not have co-relation with label.

- **Pre-Processing Steps**

1. Identifying sources of the data
2. Analysing the information
3. Cleaning and handling the information
4. Selecting the most significant elements
5. Writing down findings and observations
6. Using various models to train the data
7. Selecting the best-fitted model for predictions
8. Predicting results for test information

- **Pre-Processing Pipeline.**

- First let's check the data type of the dataset.

```
df.info()
```

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

As we can see, Workclass, Education, Marital_status, Occupation, Relationship, Race, Sex, Native_country and Income columns are Object type data which needs to change to Integer, since it is important for model building as model does not consider the string value.

- There are no Null Values present in the dataset so we can move further.

```
df.isnull().sum()
```

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

- Sometimes some unwanted things can be found in a dataset which are equivalent to Null values. It is important to take care of such cases.

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

- Used Mean and Mode method to fill those unwanted characters.
- Encoded data into Integer

```
for i in df.columns:
    if df[i].dtypes=='object':
        df[i]=enc.fit_transform(df[i].values.reshape(-1,1))
```

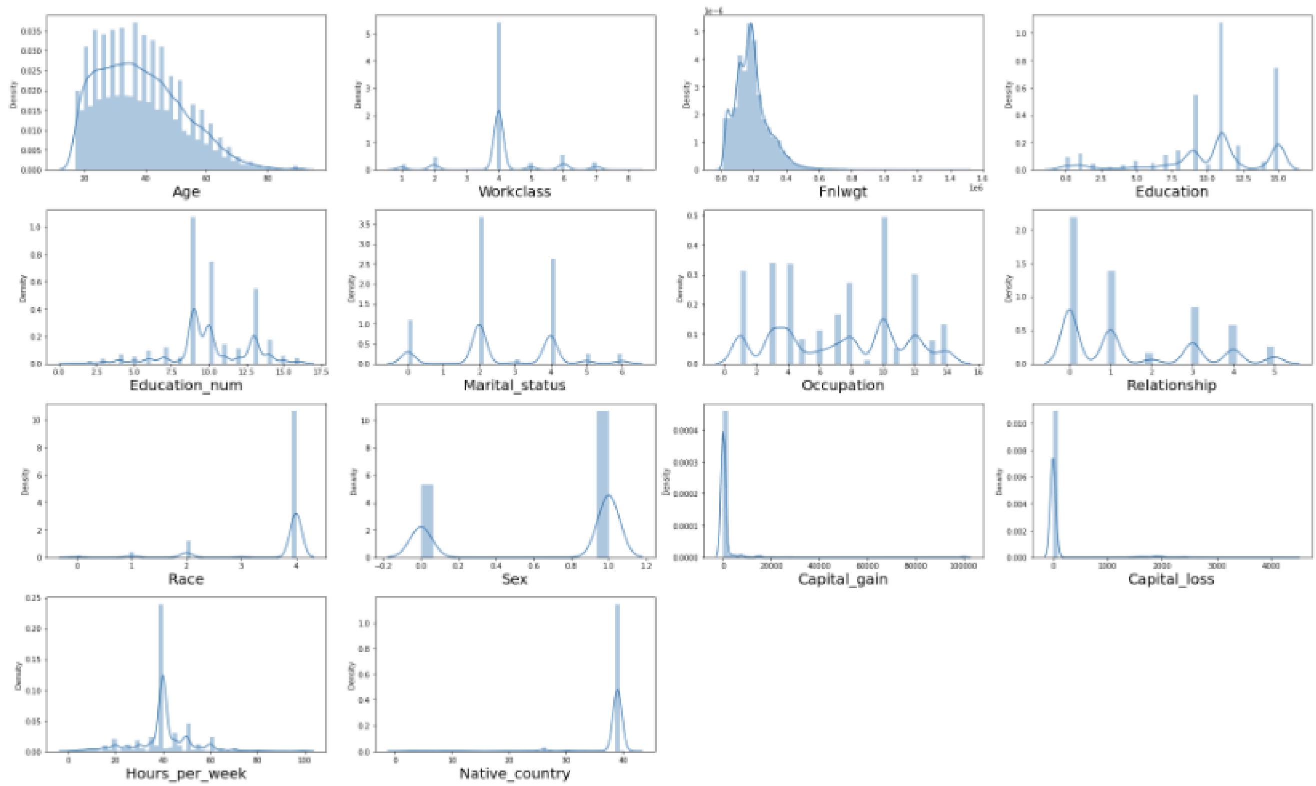
Endcoded data into Int

- Checked the skewness of the data.

```
df.skew()
```

```
Age                0.558738
Workclass          0.076178
Fnlwgt             1.446972
Education         -0.934063
Education_num     -0.311630
Marital_status    -0.013448
Occupation         0.000536
Relationship       0.786784
Race              -2.435332
Sex               -0.719244
Capital_gain      11.953690
Capital_loss       4.594549
Hours_per_week     0.227636
Native_country    -4.243083
Income             1.212383
dtype: float64
```

```
plt.figure(figsize=(25,15), facecolor='white')

plotno = 1

for column in x:
    if plotno <= 16:
        ax = plt.subplot(4,4,plotno)
        sns.distplot(x[column])
        plt.xlabel(column,fontsize=20)

    plotno+=1
plt.tight_layout()
```



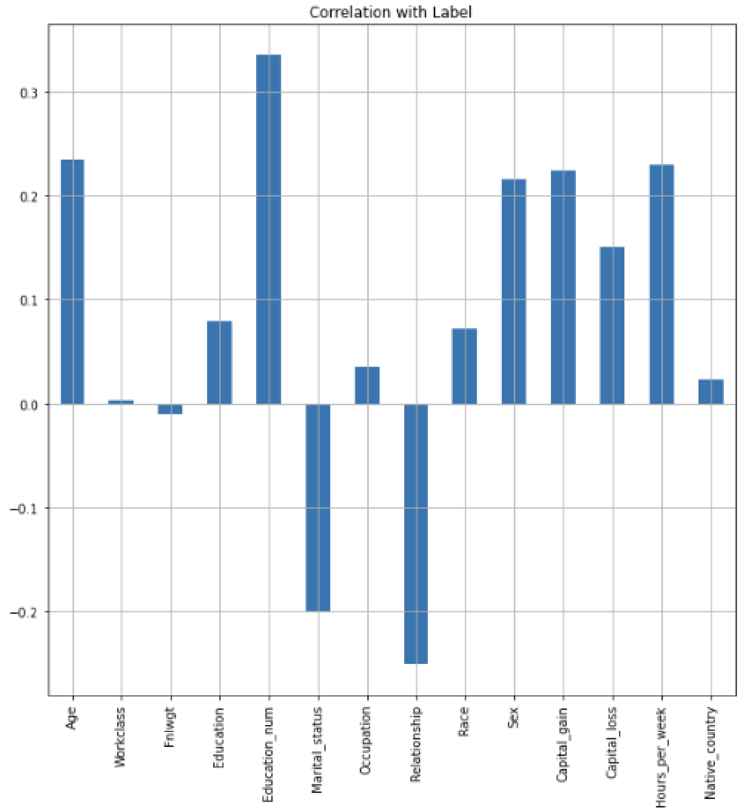
Fnlwgt, Education, Relationship, Race, Sex, Capital_gain, Capital_loss and Native_country has skewness and can have outliers.
Education, Relationship, Race, Sex and Native_country are the classified columns hence cannot remove the outliers.

- Checked the co-relation with label.

```
df.drop('Income',axis=1).corrwith(df.Income)
```

```
Age               0.234039
Workclass         0.002739
Fnlwgt           -0.009481
Education         0.079311
Education_num     0.335182
Marital_status   -0.199295
Occupation        0.034599
Relationship     -0.250924
Race              0.071853
Sex               0.215995
Capital_gain      0.223333
Capital_loss      0.150523
Hours_per_week    0.229690
Native_country    0.023063
dtype: float64
```

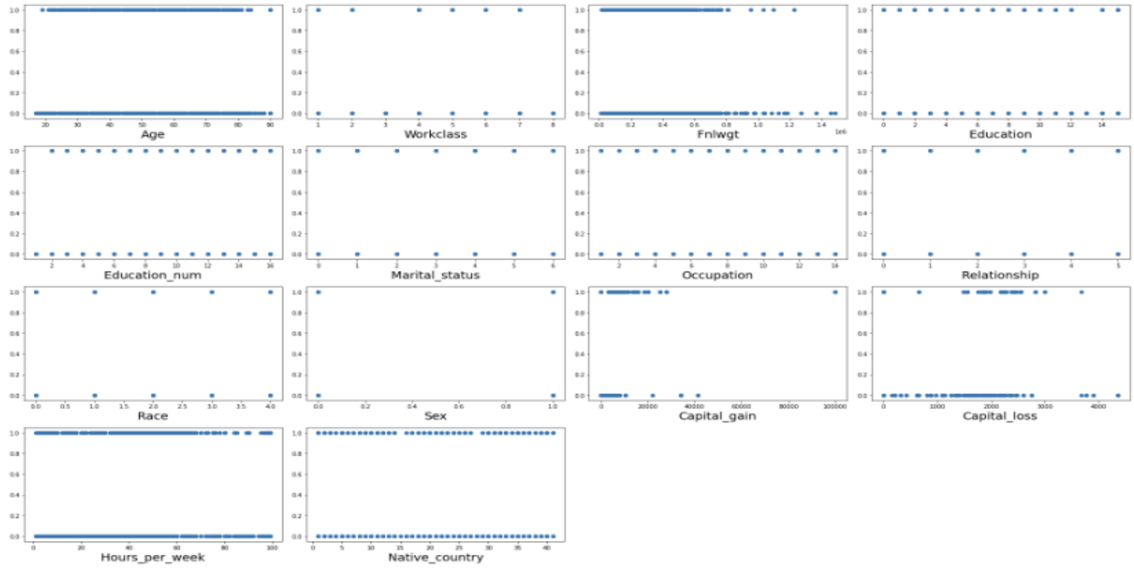
```
df.drop('Income',axis=1).corrwith(df.Income).plot(kind='bar',grid=True,figsize=(10,10),
                                                   title="Correlation with Label")
plt.show()
```



As we can observe in the dataset that 2 columns are inversely proportionate with label. Marital_status, Fnlwgt and Relationship have inverse relationship with label.

```
In [11]:
x = df.drop(['Income'],axis=1)
y = df['Income']
```

```
In [12]:
plt.figure(figsize=(25,15), facecolor='white')

plotno = 1

for column in x:
    if plotno <= 16:
        ax = plt.subplot(4,4,plotno)
        plt.scatter(x[column],y)
        plt.xlabel(column,fontsize=20)

    plotno+=1
plt.tight_layout()
```



Workclass, Education, Occupation, Race and Native_country have less or no corelation with label.

Age, Education_num, Relationship, Sex, Capital_gain and Hours_per_week have the highest co-relation with label.

Dropping Workclass, Occupation, and Native_country columns since it has no co-relation with Label.

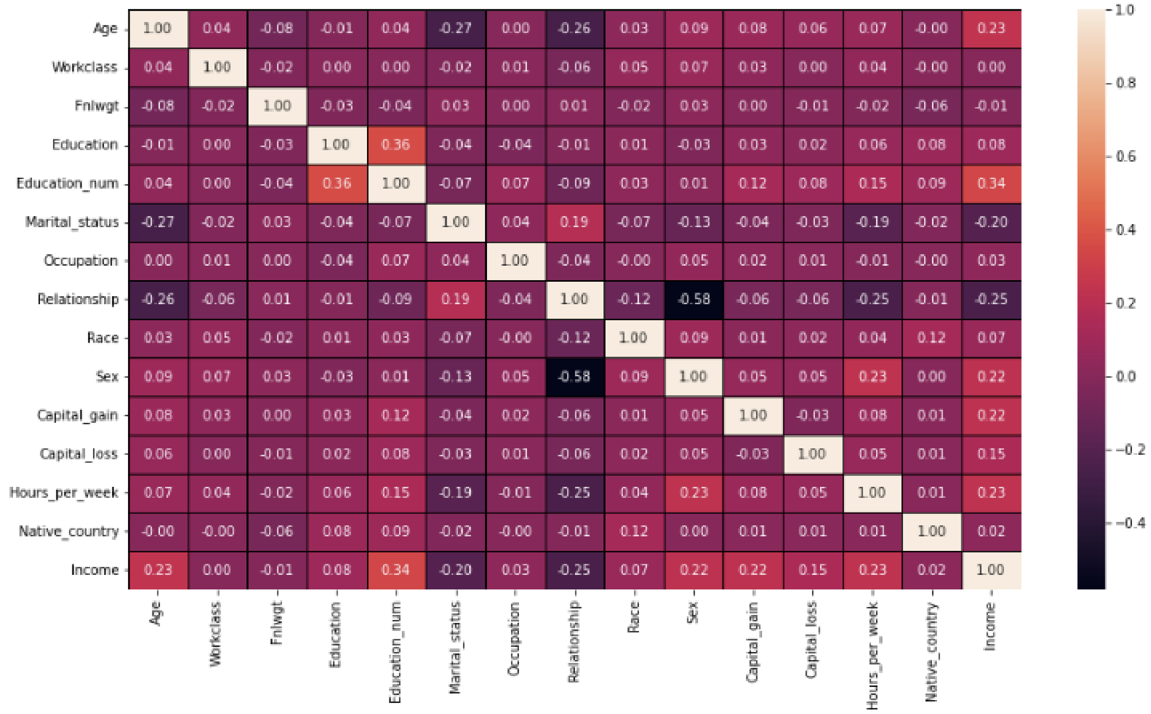
```
df = df.drop(['Workclass', 'Occupation', 'Native_country'],axis=1)
```

Findings:

1. It should be visible in the diagram that Education_num is the significant variable that influences the pay of an individual. Since advanced education can give a job to an individual.
2. Relationship is another significant element that conversely impacts the pay of an individual. In this way, we can presume that an individual who has fewer relationships will procure great.
3. Age likewise influences the pay as higher the age will in general have more experience which brings about higher pay.
4. Hours_per_week shows a number of working hours and it has a decent co-connection with pay since a higher quantity of working hours will bring about higher pay.
5. As per the dataset Sex or orientation of the individual likewise has a decent co-connection with the payment of an individual.
6. Capital_gain shows that the individual has procured any benefit and it affects the pay of an individual as more the capital increase will bring about a big-time salary.

- Checked the Multicollinearity issue in the data.

```
plt.figure(figsize = (15,8))
sns.heatmap(df.corr(),annot=True, linewidths=0.5,linecolor='black',fmt = '.2f')
plt.show()
```



Multicollinearity issue doesn't exist in this data set
Most elevated Multicollinearities exist among Education_num and Education segments i.e., 36% which is not that great.

- Checked the VIF Score of features
  Variance inflation factor (VIF) is a measure of the amount of multicollinearity in a set of multiple features or variables.

```
In [20]: x = df[['Age', 'Fnlwgt', 'Education', 'Education_num', 'Marital_status',
                 'Relationship', 'Race', 'Sex', 'Capital_gain', 'Capital_loss',
                 'Hours_per_week']]
```

```
In [21]: vif = pd.DataFrame()
         vif["Features"] = x.columns
```

```
In [22]: vif["VIF"] = [variance_inflation_factor(x.values, i) for i in range(len(x.columns))]
```

```
In [23]: vif
```

Out[23]:

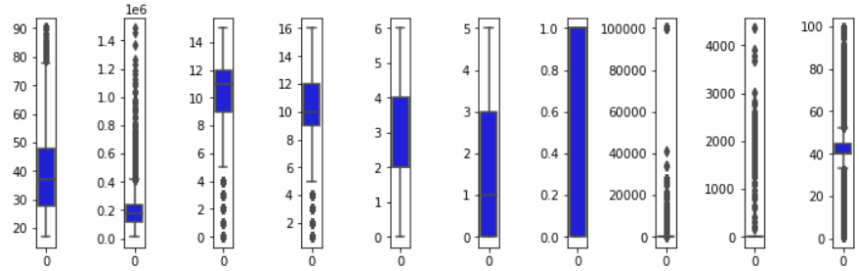
	Features	VIF
0	Age	8.107316
1	Fnlwgt	4.007132
2	Education	9.050872
3	Education_num	17.325276
4	Marital_status	3.869540
5	Relationship	2.506080
6	Race	15.083497
7	Sex	4.334902
8	Capital_gain	1.043199
9	Capital_loss	1.061353
10	Hours_per_week	11.460027

Race and Education_num column has the highest VIF, however Race has low co-relation with Label hence dropping Race column.

```
df = df.drop(['Race'],axis=1)
```

- Checked the outliers in the data

```
a = x.columns.values
col = 30
row = 14
plt.figure(figsize = (col,3*row))
for i in range(0, len(a)):
    plt.subplot(row,col,i+1)
    sns.boxplot(data = x[a[i]],color='blue',orient='v')
    plt.tight_layout()
```



1. Outliers are present in Age, Fnlwgt, Education, Education_num, Hours_per_week, Capital_loss and Capital_gain
2. Education and Education_num are classified columns hence not removing outliers.

- Removed the Outliers
  Data Loss due to removing outliers is 9804

```
data_loss = old_data - new_data
print('Lost', data_loss,'no. of Data')
```

```
Lost 9804 no. of Data
```

- Used power transformation to remove skewness from the data.

```
scaler = PowerTransformer(method='yeo-johnson')
```

```
df[['Age','Fnlwgt','Hours_per_week','Capital_loss','Capital_gain']] = scaler.fit_transform(df[['Age','Fnlwgt','Hours_per_wee
```

- Label data was not balance hence have used SMOTE technique to balance the data after scaling the data

In [44]:
```
scaler = StandardScaler()
X_scale = scaler.fit_transform(x)
```

Scaling the data

In [45]:
```
x_train,x_test,y_train,y_test = train_test_split(X_scale,y,test_size = 0.01,random_state = 65)
```

Have added test_size small so that we can not loose train data

In [46]:
```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
```

In [47]:
```
ove_smp=SMOTE(0.75)

x_train_new, y_train_new = ove_smp.fit_sample(x_train, y_train)
```

In [48]:
```
print (y_train.value_counts())
print (y_train_new.value_counts())
```

```
0.0    16752
1.0     5776
Name: Income, dtype: int64
0.0    16752
1.0    12564
Name: Income, dtype: int64
```

- **EDA Concluding Remark.**

1. The information was not organized and coordinated and subsequently cleaned the information utilizing different information cleaning and pre-handling techniques.
2. There are numerous anomalies present in the information consequently eliminating exceptions
3. There was a skewness in the information thus have eliminated the skewness from the information.
4. There was an irregularity in the information thus have utilized SMOTE strategy to balance the information.
5. Scaled the data utilizing Standard Scalar to make the information normalized to fabricate a model.

- **Hardware and Software Requirements and Tools Used**
  1. Libraries and packages used
  - import numpy as np - For Numpy work
  - import pandas as pd - To work on DataFrame
  - import seaborn as sns - Plotting Graphs
  - import matplotlib.pyplot as plt - Plotting Graphs
  - import pickle – To save the Model
  - from sklearn.preprocessing import StandardScaler (To scale the train data), OrdinalEncoder(To encode object data to Integer), PowerTransformer (To remove skewness from dataset)
  - from statsmodels.stats.outliers_influence import variance_inflation_factor
  - enc = OrdinalEncoder() = Assigned OrdinalEncoder to variable
  - from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score,(To split the data into train and test, Search the best parameters, to calculate cross validation score)
  - from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, roc_auc_score - To calculate and analyse model metrics.
  - from sklearn import metrics

  Models which are used
  - from sklearn.ensemble import RandomForestClassifier
  - from sklearn.linear_model import LogisticRegression
  - from sklearn.ensemble import GradientBoostingClassifier
  - from sklearn.tree import DecisionTreeClassifier
  - from sklearn.neighbors import KNeighborsClassifier
  - from sklearn.svm import SVC

  - import warnings
  - warnings.filterwarnings('ignore') - To ignore unwanted Warnings

  2. Hardware used – 11th Gen Intel(R) Core (TM) i3-1115G4 @ 3.00GHz 3.00 GHz with 8.00 GB RAM and Windows 11
  3. Software used – Anaconda and Jupyter Notebook to build the model.

- ## **Building Machine Learning Models.**

I have built 6 machine learning models to predict the label. Below are the machine learning models which are been used.

1. LogisticsRegression
2. RandomForestClassifier
3. DecisionTreeClassifier
4. GradientBoostingClassifier
5. Support Vector Classifier
6. KNeighborsClassifier

1. LogisticsRegression:

Have used "For Loop" to find out the highest accuracy score with different random state ranging from 0-100 and using that random state to split the data into train and test data.

```
In [50]:
maxAccu =0
maxRS= 0

for i in range(1,200):
    x_train,x_test,y_train,y_test = train_test_split(X_scale,y,test_size = 0.25,random_state = i)
    log = LogisticRegression()
    log.fit(x_train,y_train)
    y_pred=log.predict(x_test)
    acc=accuracy_score(y_test, y_pred)
    print('accuracy', acc,'Random_state',i)

    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
        print('max_accuracy', maxAccu,'max_Random_state',i)
```

```
accuracy 0.7627234274798744 Random_state 169
accuracy 0.761086096329649З Random_state 170
accuracy 0.7676354209305498 Random_state 171
accuracy 0.767498976668031l Random_state 172
accuracy 0.7722745258561877 Random_state 173
accuracy 0.7612225405921681 Random_state 174
accuracy 0.7696820848683313 Random_state 175
accuracy 0.7516714422158548 Random_state 176
accuracy 0.7758220766816756 Random_state 177
max_accuracy 0.7758220766816756 max_Random_state 177
accuracy 0.7526265520534862 Random_state 178
accuracy 0.753854550416155 Random_state 179
accuracy 0.7580843225542366 Random_state 180
accuracy 0.762996316004912 Random_state 181

accuracy 0.7632692045299495 Random_state 182
accuracy 0.7702278619184063 Random_state 183
accuracy 0.7616318733797244 Random_state 184
accuracy 0.7703643061809251 Random_state 185
accuracy 0.7583572110792741 Random_state 186
```

```
In [79]:
x_train,x_test,y_train,y_test = train_test_split(X_scale,y,test_size = 0.25,random_state = 177)
```

Have used "Define Function" to define a machine learning model code that automatically provides the train and test accuracy code.

Formulas:

y_pred = clf.predict(x_train) = Predicting train data

accuracy_score(y_train, y_pred) = Calculating train accuracy score (comparing y_pred data with y_train data)

pred = clf.predict(x_test) = Predicting test data

accuracy_score(y_test, pred) = Calculating test accuracy score (comparing pred data with y_test data)

```python
def print_score(clf, x_train,x_test,y_train,y_test, train=True):
    if train:
        y_pred = clf.predict(x_train)

        print('\n===============Train Result===============')
        print(f'Accuracy Score: {accuracy_score(y_train, y_pred)*100:.2f}%')


    elif train==False:
        pred = clf.predict(x_test)

        print('\n===============Test Result===============')
        print(f'Accuracy Score: {accuracy_score(y_test, pred)*100:.2f}%')

        print('\n \n Test Classification Report \n', classification_report(y_test, pred, digits=2))

        scr_log = cross_val_score(clf,X_scale,y,cv=5)
        print('Cross Validation Score- ', scr_log.mean())
```

Trained the data and run the "Def" function

```
In [81]:
log = LogisticRegression()
log.fit(x_train,y_train)

print_score(log,x_train,x_test,y_train,y_test, train=True)
print_score(log,x_train,x_test,y_train,y_test, train=False)
```

```
===============Train Result===============
Accuracy Score: 75.97%

===============Test Result===============
Accuracy Score: 77.58%


 Test Classification Report
               precision    recall  f1-score   support

         0.0       0.79      0.83      0.81      4199
         1.0       0.75      0.71      0.73      3130

    accuracy                           0.78      7329
   macro avg       0.77      0.77      0.77      7329
weighted avg       0.77      0.78      0.77      7329

Cross Validation Score-  0.7616660508160524
```

2. RandomForestClassifier:

In [82]:
```
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)

print_score(rfc,x_train,x_test,y_train,y_test, train=True)
print_score(rfc,x_train,x_test,y_train,y_test, train=False)
```

```
===============Train Result===============
Accuracy Score: 99.94%

===============Test Result===============
Accuracy Score: 85.51%


 Test Classification Report
              precision    recall  f1-score   support

         0.0       0.88      0.87      0.87      4199
         1.0       0.82      0.84      0.83      3130

    accuracy                           0.86      7329
   macro avg       0.85      0.85      0.85      7329
weighted avg       0.86      0.86      0.86      7329

Cross Validation Score-  0.8550975328202227
```

3. DecisionTreeClassifier:

In [83]:
```
dtc = DecisionTreeClassifier()
dtc.fit(x_train,y_train)

print_score(dtc,x_train,x_test,y_train,y_test, train=True)
print_score(dtc,x_train,x_test,y_train,y_test, train=False)
```

```
===============Train Result===============
Accuracy Score: 99.94%

===============Test Result===============
Accuracy Score: 81.96%


 Test Classification Report
              precision    recall  f1-score   support

         0.0       0.84      0.84      0.84      4199
         1.0       0.79      0.79      0.79      3130

    accuracy                           0.82      7329
   macro avg       0.82      0.82      0.82      7329
weighted avg       0.82      0.82      0.82      7329

Cross Validation Score-  0.8117763047520474
```

4. GradientBoostingClassifier:

In [84]:
```
gbdt = GradientBoostingClassifier()
gbdt.fit(x_train,y_train)

print_score(gbdt,x_train,x_test,y_train,y_test, train=True)
print_score(gbdt,x_train,x_test,y_train,y_test, train=False)
```

```
===============Train Result===============
Accuracy Score: 83.45%

===============Test Result===============
Accuracy Score: 83.59%


 Test Classification Report
              precision    recall  f1-score   support

         0.0       0.87      0.84      0.85      4199
         1.0       0.80      0.83      0.81      3130

    accuracy                           0.84      7329
   macro avg       0.83      0.83      0.83      7329
weighted avg       0.84      0.84      0.84      7329

Cross Validation Score-  0.8282512200473802
```

5. Support Vector Classifier:

In [85]:
```
svc = SVC()
svc.fit(x_train,y_train)

print_score(svc,x_train,x_test,y_train,y_test, train=True)
print_score(svc,x_train,x_test,y_train,y_test, train=False)
```

```
===============Train Result===============
Accuracy Score: 80.72%

===============Test Result===============
Accuracy Score: 80.64%


 Test Classification Report
              precision    recall  f1-score   support

         0.0       0.87      0.78      0.82      4199
         1.0       0.74      0.85      0.79      3130

    accuracy                           0.81      7329
   macro avg       0.80      0.81      0.81      7329
weighted avg       0.81      0.81      0.81      7329

Cross Validation Score-  0.8030769067886826
```

6. KNeighborsClassifier:

In [86]:
```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)

print_score(knn,x_train,x_test,y_train,y_test, train=True)
print_score(knn,x_train,x_test,y_train,y_test, train=False)
```

```
===============Train Result===============
Accuracy Score: 87.24%

===============Test Result===============
Accuracy Score: 81.44%


 Test Classification Report
              precision    recall  f1-score   support

         0.0       0.88      0.79      0.83      4199
         1.0       0.75      0.85      0.80      3130

    accuracy                           0.81      7329
   macro avg       0.81      0.82      0.81      7329
weighted avg       0.82      0.81      0.82      7329

Cross Validation Score-  0.8220774126548923
```

- **Findings**

  - **LogisticsRegression** - Cross Validation Score is 76.16%, Accuracy Score of Train Result is 75.97% and Test Result is 77.58%
  - **RandomForestClassifier** - Cross Validation Score is 85.50%, Accuracy Score of Train Result is 99.94% and Test Result is 85.51%
  - **DecisionTreeClassifier** - Cross Validation Score is 81.17%, Accuracy Score of Train Result is 99.94% and Test Result is 81.96%
  - **GradientBoostingClassifier** - Cross Validation Score is 82.82%, Accuracy Score of Train Result is 83.45% and Test Result is 83.59%
  - **Support Vector Classifier** - Cross Validation Score is 80.30%, Accuracy Score of Train Result is 80.72% and Test Result is 80.64%
  - **KNeighborsClassifier** - Cross Validation Score is 82.20%, Accuracy Score of Train Result is 87.24% and Test Result is 81.44%

- **Model Selection:**

  Selecting GradientBoostingClassifier as it has low variance between train and test result and has high accuracy i.e., 83.45% and 83.59% respectively.

- **Hyper Parameter Tuning:**

  In machine learning, hyperparameter tuning is the issue of picking a bunch of ideal hyperparameters for a learning calculation. hyperparameter tuning boundaries rely upon the choice of the model, as the model changes the parameters additionally change. A hyperparameter is a parameter whose worth is utilized to control the learning experience. According to the choice of the GradientBoostingClassifier model, we should tune the model to expand the train and test precision.

  Have referenced the parameters and utilized Grid Search CV to find the best mix of parameters that will build the precision.

```
In [67]:
grid_param = {
    'loss': ['deviance', 'exponential'],
    'learning_rate': np.arange(0.1,0.9,0.1),
    'criterion':['friedman_mse', 'mse', 'mae'],
}
```

Selecting Parameters for Hyper Parameter Tuning

```
In [68]:
grid_search = GridSearchCV(estimator=gbdt,
                           param_grid=grid_param,
                           cv=3,
                           n_jobs =-1)
```

Searching the best grid for the model

```
In [69]:
grid_search.fit(x_train,y_train)

Out[69]: GridSearchCV(cv=3,
             estimator=GradientBoostingClassifier(criterion='mse',
                                                  learning_rate=0.5),
             n_jobs=-1,
             param_grid={'criterion': ['friedman_mse', 'mse', 'mae'],
                         'learning_rate': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]),
                         'loss': ['deviance', 'exponential']})
```

Have not used more parameters as they take more time to train and due to slow hardware, it gets impossible to train the model.

Used best fitted parameters to train the model.

```
In [70]:
best_parameters = grid_search.best_params_
print(best_parameters)

{'criterion': 'friedman_mse', 'learning_rate': 0.8, 'loss': 'deviance'}
```

```
In [74]:
gbdt = GradientBoostingClassifier(criterion='friedman_mse', learning_rate=0.8, loss='deviance')
gbdt.fit(x_train,y_train)

Out[74]: GradientBoostingClassifier(learning_rate=0.8)
```

Model after Hyper Tuning:

```
In [75]:
y_pred = gbdt.predict(x_test)
pred = gbdt.predict(x_train)
```

```
In [76]:
print(f'Train Accuracy Score: {accuracy_score(y_train, pred)*100:.2f}%')
print(f'Test Accuracy Score: {accuracy_score(y_test, y_pred)*100:.2f}%')
print(classification_report(y_test, y_pred))
```

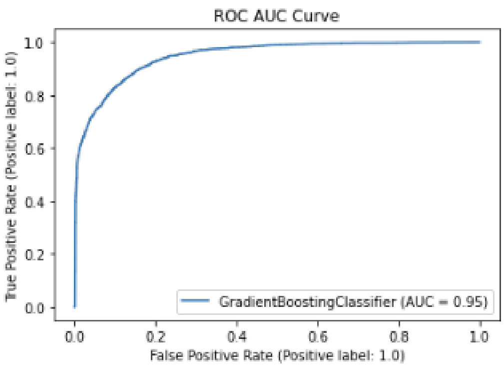
```
Train Accuracy Score: 88.67%
Test Accuracy Score: 86.97%
              precision    recall  f1-score   support

         0.0       0.88      0.90      0.89      4227
         1.0       0.86      0.83      0.84      3102

    accuracy                           0.87      7329
   macro avg       0.87      0.86      0.87      7329
weighted avg       0.87      0.87      0.87      7329
```

- Previous Train accuracy score was 83.45% and new Train accuracy score is 88.67%
- Previous Test accuracy score was 83.59% and new Test accuracy score is 86.97%

- **ROC AUC Curve:**

  AUC – ROC (Area Under Curve - Receiver operating characteristic) curve is a performance measurement for the classification issues at different limit settings. ROC is a probability curve and AUC addresses the degree or proportion of detachability. It tells how much the model is fit for recognizing classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By relationship, the Higher the AUC, the better the model is at recognizing income above $50K and income below $50K.

```
In [77]:
from sklearn.metrics import plot_roc_curve
plot_roc_curve(gbdt,x_test,y_test)
plt.title("ROC AUC Curve")
plt.show()
```



AUC score is 95% which is pretty good.

- # Concluding Remarks.
- ## Key Findings and Conclusions of the Study

1. Selecting GradientBoostingClassifier model since the Accuracy score i.e., 83.45% and test scores i.e., 83.59% are greater and close to each other.
2. After tuning the model the train and test accuracy score increased to 5% in train data and 3% in test data.
3. AUC score is also high i.e 95%.

- **Saving the Model**

Saving the selected model after hyper parameter using pickle.

```
In [87]:  file = 'Census_Income_Project.pickle'
          pickle.dump(gbdt, open(file, 'wb'))
```

- **Learning Outcomes of the Study in respect of Data Science**

1. Data Cleaning assists with changing over sloppy and unstructured data into organized data which will be utilized to make discoveries.
2. Data visualization gets it and dissects the information.
3. Model structure assists with anticipating results, for this situation GradientBoostingClassifier model fits ideal for this dataset.
4. This model can be utilized in different use cases like publicizing, marketing, research, lead generation, advertising, promoting, finance, etc.

- **Limitations of this work and Scope for Future Work**

1. There are 14 features present in the dataset anyway due to pre-handling and representation we have cut down a few elements, consequently this could turn into a disadvantage in the future as we update the dataset and there is the possibility that we might lose some significant data.
2. It is important to watch out for new and refreshed information to further train the model and settle on choices according to new information.