Census Income Project Using Python

> *TODO- Introducing ML project for census income. In this blog-post, I will go through the whole process of creating a machine learning model on the census income dataset.*

Problem Statement:

This data was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics). A set of reasonably clean records was extracted using the following conditions: ((AAGE>16) & (AGI>100) & (AFNLWGT>1) && (HRSWK>0)). ***The prediction task is to determine whether a person makes over $50K a year or less then.***

Here's a step-by-step outline of the project:

1. Importing necessary libraries.
2. Importing dataset from GitHub.
3. Exploratory Data Analysis (EDA).
4. Data Preprocessing & Feature Engineering.
5. Model building and Saving.

Importing necessary libraries :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from plotly.offline import iplot
import plotly as py
py.offline.init_notebook_mode(connected=True)


from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report


import warnings
warnings.filterwarnings('ignore')
```

Importing dataset from GitHub.

```
df_census =
pd.read_csv("https://raw.githubusercontent.com/dsrscientist/
dataset1/master/census_income.csv")
```

Exploratory Data Analysis (EDA).

```
df_census.head()
```

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship	Race
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White

Sex	Capital_gain	Capital_loss	Hours_per_week	Native_country	Income
Male	0	0	13	United-States	<=50K
Male	0	0	40	United-States	<=50K
Male	0	0	40	United-States	<=50K
Female	0	0	40	Cuba	<=50K
Female	0	0	40	United-States	<=50K

Through the head().method we can observed 5 rows. Here we can see there are many features which are in categorical so we have to convert into numeric in order to feed our data to machine learning model. As per our problem statement we have to predict the income is below 50k or beyond. So we can convert into binary and build classification model.

```
df_census.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
```

The data-set contains 32560 **rows and** 14 **features + the target variable (**Income**)**. 6 are integers and 9 are objects. Below I have listed the features with a short description:

```
0   Age             : The age of people
 1   Workclass      : Type of job
 2   Fnlwgt         : Final weight
 3   Education      : Education status
 4   Education_num  : Number of years of education in total
 5   Marital_status : Marital_status
 6   Occupation     : Occupation
 7   Relationship   : Relationship
 8   Race           : residential segregation
 9   Sex            : Gender
 10  Capital_gain   : Capital gain is the profit one earns
 11  Capital_loss   : Capital gain is the profit one loose
 12  Hours_per_week : Earning rate as per hrs
 13  Native_country : Country
 14  Income         : Income (Target variable)


df_census.describe()
```

	Age	Fnlwgt	Education_num	Capital_gain	Capital_loss	Hours_per_week
count	32560.000000	3.256000e+04	32560.000000	32560.000000	32560.000000	32560.000000
mean	38.581634	1.897818e+05	10.080590	1077.615172	87.306511	40.437469
std	13.640642	1.055498e+05	2.572709	7385.402999	402.966116	12.347618
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178315e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783630e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370545e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

As per above sort description we have seen in ‘count’ there is no null values in the dataset. And, we can also see ‘mean’ it shows variation among the features and values are on different scales so we have to scale the features in similar scale.

As we have checked statistical descriptions which shows only numeric data and dataset contains categorical values as well as. Therefore we have to check null values.

```
df_census.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
```

As per above we can not see any null values in our dataset. But in the 2 features Native_country and Occupation there is '?' empty which will be consider as a null values. Hence we have to replace with authentic values.

Checking the unique values of both the columns.

```
df_census['Native_country'].unique()
```

```
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)
```

Here we can see the "?" missing values.

```
df_census['Occupation'].unique()
```

```
array([' Exec-managerial', ' Handlers-cleaners', ' Prof-specialty',
       ' Other-service', ' Adm-clerical', ' Sales', ' Craft-repair',
       ' Transport-moving', ' Farming-fishing', ' Machine-op-inspct',
       ' Tech-support', ' ?', ' Protective-serv', ' Armed-Forces',
       ' Priv-house-serv'], dtype=object)
```

Here we can see the "?" missing values.

```
df_census.loc[df_census.Native_country==' ?']
```

Out[17]:

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship	Race	Sex	Capital_gain	Capital_loss	Hours_per_we
13	40	Private	121772	Assoc-voc	11	Married-civ-spouse	Craft-repair	Husband	Asian-Pac-Islander	Male	0	0	
37	31	Private	84154	Some-college	10	Married-civ-spouse	Sales	Husband	White	Male	0	0	
50	18	Private	226956	HS-grad	9	Never-married	Other-service	Own-child	White	Female	0	0	
60	32	?	293936	7th-8th	4	Married-spouse-absent	?	Not-in-family	White	Male	0	0	
92	30	Private	117747	HS-grad	9	Married-civ-spouse	Sales	Wife	Asian-Pac-Islander	Female	0	1573	
...	...	...	...	...	...	...	...	...	...	...	...	...	
32448	44	Self-emp-inc	71556	Masters	14	Married-civ-spouse	Sales	Husband	White	Male	0	0	
32468	58	Self-emp-inc	181974	Doctorate	16	Never-married	Prof-specialty	Not-in-family	White	Female	0	0	
32491	42	Self-emp-not-inc	217597	HS-grad	9	Divorced	Sales	Own-child	White	Male	0	0	
32509	39	Private	107302	HS-grad	9	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	
32524	81	?	120478	Assoc-voc	11	Divorced	?	Unmarried	White	Female	0	0	

583 rows × 15 columns

```
df_census.loc[df_census.Occupation==' ?']
```

Out[23]:

	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship	Race	Sex	Capital_gain	Capital_loss	Hours_per_we
26	54	?	180211	Some-college	10	Married-civ-spouse	?	Husband	Asian-Pac-Islander	Male	0	0	
68	25	?	200681	Some-college	10	Never-married	?	Own-child	White	Male	0	0	
76	67	?	212759	10th	6	Married-civ-spouse	?	Husband	White	Male	0	0	
105	17	?	304873	10th	6	Never-married	?	Own-child	White	Female	34095	0	
127	35	?	129305	HS-grad	9	Married-civ-spouse	?	Husband	White	Male	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
32529	35	?	320084	Bachelors	13	Married-civ-spouse	?	Wife	White	Female	0	0	
32530	30	?	33811	Bachelors	13	Never-married	?	Not-in-family	Asian-Pac-Islander	Female	0	0	
32538	71	?	287372	Doctorate	16	Married-civ-spouse	?	Husband	White	Male	0	0	
32540	41	?	202822	HS-grad	9	Separated	?	Not-in-family	Black	Female	0	0	
32541	72	?	129912	HS-grad	9	Married-civ-spouse	?	Husband	White	Male	0	0	

1816 rows × 15 columns

In the above observations we can see “Native_country” contains 583 missing values and “Occupation” contains 1816 missing values as count is huge. Although we can use simple imputer or other method like KNN imputer to fill null values. But if we will fill it probably it may biased towards the single variable. However, we have large data so we can remove it using ‘Drop’ function.

```
df_census.drop(df_census[df_census['Native_country'] == '
?'].index,inplace=True)


# droping the bull values from both the columns


df_census.drop(df_census[df_census['Occupation'] == '
?'].index,inplace=True)
```

Now checking the unique value count again it has removed or not

```
df_census['Native_country'].unique()


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)


df_census.drop(df_census[df_census['Occupation'] == '
?'].index,inplace=True)  # removeing the unnamed values


array([' Bachelors', ' HS-grad', ' 11th', ' Masters', '
9th',
       ' Some-college', ' Assoc-acdm', ' 7th-8th', '
Doctorate',
       ' Assoc-voc', ' Prof-school', ' 5th-6th', ' 10th', '
```

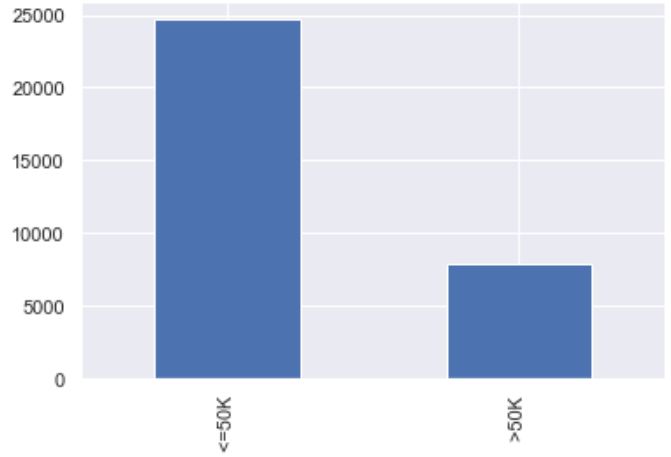
```
Preschool',
       ' 12th', ' 1st-4th'], dtype=object)
```

We can see above values where “?” has been removed.

Let’s take a more detailed look from Data Visualizations:

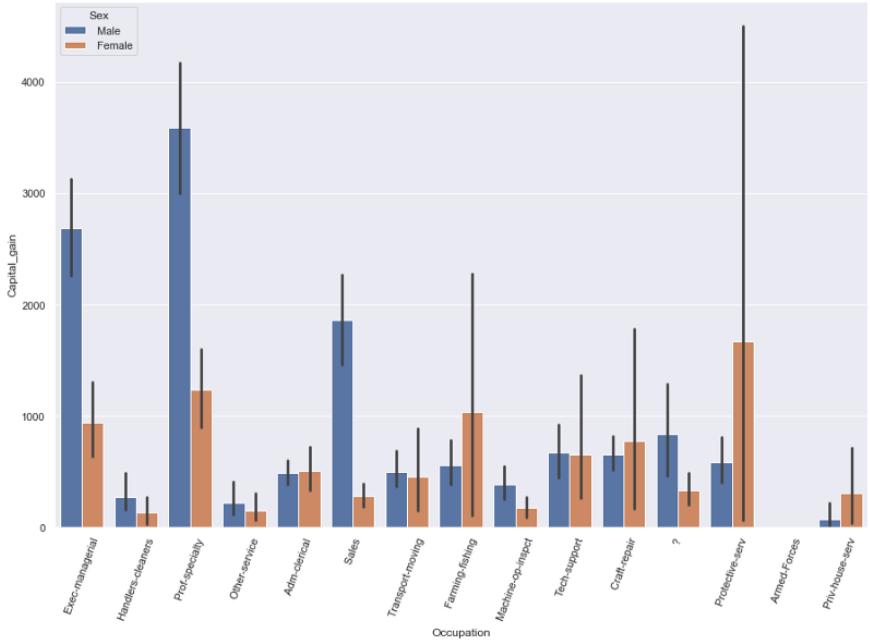
```
df_census['Income'].value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d549ccbe20>
```



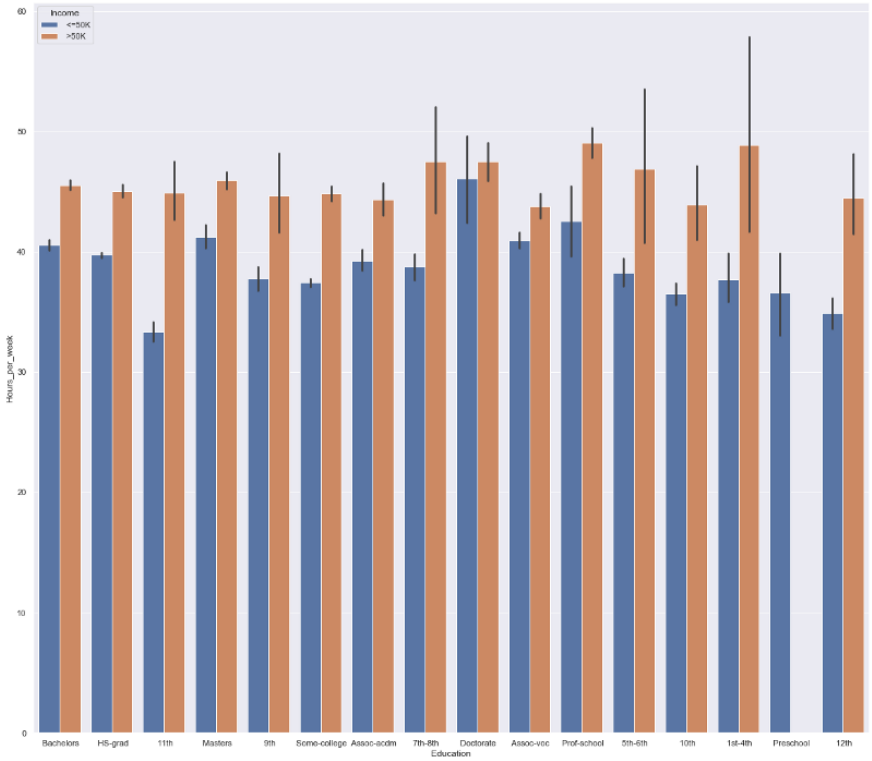
As per above visualization of target variable we can see the ratio is imbalanced.

```
plt.figure(figsize=(15,10))
sns.barplot(x='Occupation',y='Capital_gain',data=df_census,h
ue='Sex')
plt.xticks(rotation=70)


##Checking the distribution of Capital_gain and  Occupation
as per gender##
```




```
plt.figure(figsize=(20,18))
sns.barplot(x='Education',y='Hours_per_week',data=df_census,
hue='Income')
```

##Checking the distribution of Education and Hours_per_week as gender###



Data Preprocessing & Feature Engineering :

Encoding Categorical Variables :

As we have many features contains categorical variable so we are using pandas get_dummies function to convert into numeric and 2 variables “Income” and “Sex” columns conerting into binary using label encoder.

```
le = LabelEncoder() # label encoder

df_census['Income']=le.fit_transform(df_census['Income'])
df_census['Sex']=le.fit_transform(df_census['Sex'])

#Converting 2 columns into binary

df_census = pd.get_dummies(df_census,drop_first=True)

pd.set_option('display.max_columns',100)#to display all
columns
```

Now our data set has been transform into numeric.

```
df_census.head()
```

	Age	Fnlwgt	Education_num	Sex	Capital_gain	Capital_loss	Hours_per_week	Income	Workclass_ Local-gov	Workclass_ Private	Workclass_ Self-emp-inc	Workclass_ Self-emp-not-inc	Workclass_ State-gov
0	50	83311	13	1	0	0	13	0	0	0	0	1	0
1	38	215646	9	1	0	0	40	0	0	1	0	0	0
2	53	234721	7	1	0	0	40	0	0	1	0	0	0
3	28	338409	13	0	0	0	40	0	0	1	0	0	0
4	37	284582	14	0	0	0	40	0	0	1	0	0	0

Workclass_ Self-emp-inc	Workclass_ Self-emp-not-inc	Workclass_ State-gov	Workclass_ Without-pay	Education_ 11th	Education_ 12th	Education_ 1st-4th	Education_ 5th-6th	Education_ 7th-8th	Education_ 9th	Education_ Assoc-acdm	Education_ Assoc-voc
0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Native_country_ Scotland	Native_country_ South	Native_country_ Taiwan	Native_country_ Thailand	Native_country_ Trinadad&Tobago	Native_country_ United-States	Native_country_ Vietnam	Native_country_ Yugoslavia
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0

Now our almost data values is 0 and 1 except few features like *"'Age','Fnlwgt','Education_num','Hours_per_week'" we can use standard scaler we and convert those features in same scale.*

```
scaler = StandardScaler()


train_col_sacle =
df_census[['Age','Fnlwgt','Education_num','Hours_per_week']]


train_scaler_col = scaler.fit_transform(train_col_sacle)


train_scaler_col =
pd.DataFrame(train_scaler_col,columns=train_col_sacle.column
s)



df_census['Age']= train_scaler_col['Age']
df_census['Fnlwgt']= train_scaler_col['Fnlwgt']
df_census['Education_num']=
```

```
train_scaler_col['Education_num']
df_census['Hours_per_week']=
train_scaler_col['Hours_per_week']
```

As we have to scale only 4 features so I am stored it in a variable. And using standard scaler. later all 4 scale features replaced. Now our dataset has been scaled and ready for the model building.

Model Building And Saving.

> *Now we will train several Machine Learning models and compare their results. Note that because the dataset does not provide labels for their testing-set, we need to use the sklearn train test split and divide into test and train. Before that we have to divide into dependent and independent variables.*

```
X = df_census.drop(['Income'],axis=1)
y = df_census['Income']
```

Data is now divided in independent and dependent.

```
X_train, X_test, y_train, y_test  = train_test_split(X,y,
test_size=0.30, random_state=11)
```

Our data set divided into train and test. Now we will continue with model building.

```
lr=LogisticRegression() #Logistic Regression
knn=KNeighborsClassifier() #KNearest Neibour
dt=DecisionTreeClassifier() # Deciesion Tree
rf=RandomForestClassifier() # Random Forest
adb=AdaBoostClassifier()    # Adaboost Classifier
svm=SVC()              # support vactor classifier
gdboost=GradientBoostingClassifier() #Gradient Boosting
Classifier
xgboost=XGBClassifier()  #Xtrim Gredient Boosting Classifier


print("Model is created")
```

We are using almost 8 models. Now its time to train out model.

```
lr.fit(X_train, y_train)
knn.fit(X_train,y_train)
dt.fit(X_train,y_train)
rf.fit(X_train,y_train)
adb.fit(X_train,y_train)
svm.fit(X_train,y_train)
gdboost.fit(X_train,y_train)
xgboost.fit(X_train,y_train)
print("Model is trained")
```

Our Model has been train now checking the accuracy:

```
print("Lr classification score",lr.score(X_train,y_train))
print("knn classification score",knn.score(X_train,y_train))
print("dt classification score",dt.score(X_train,y_train))
print("rf classification score",rf.score(X_train,y_train))
print("adb classification score",adb.score(X_train,y_train))
print("svm classification score",svm.score(X_train,y_train))
print("gdboost classification
score",gdboost.score(X_train,y_train))
print("xgboost classification
score",xgboost.score(X_train,y_train))


Lr classification score 0.8516483516483516
knn classification score 0.8714948844259189
dt classification score 1.0
rf classification score 0.9999526335733232
adb classification score 0.8606479727169383
svm classification score 0.8598427434634331
gdboost classification score 0.8682266009852216
xgboost classification score 0.9067355058734369
```

Model Evaluation :

Using confusion matrix in order to evaluate the models accuracy.

```
lr_yprad = lr.predict(X_test)
knn_yprad = knn.predict(X_test)
dt_yprad = dt.predict(X_test)
```

```
rf_yprad = rf.predict(X_test)
adb_yprad = adb.predict(X_test)
svm_yprad = svm.predict(X_test)
gdboost_yprad = gdboost.predict(X_test)
xgboost_yprad = xgboost.predict(X_test)
```

Using confusion matrix in order to evaluate the models accuracy.

```
lr_conf_mat = confusion_matrix(y_test,lr_yprad)
print("confusion matrix for lr_model",'\n',lr_conf_mat)
```

```
confusion matrix for lr_model
 [[6276  544]
 [ 870 1359]]
```

```
knn_conf_mat = confusion_matrix(y_test,knn_yprad)
print("confusion matrix for knn_model",'\n',knn_conf_mat)
```

```
confusion matrix for knn_model
 [[6053  767]
 [ 930 1299]]
```

```
dt_conf_mat = confusion_matrix(y_test,dt_yprad)
print("confusion matrix for dt_model",'\n',dt_conf_mat)
```

```
confusion matrix for dt_model
 [[5895  925]
 [ 848 1381]]
```

```
rf_conf_mat = confusion_matrix(y_test,rf_yprad)
print("confusion matrix for lr_model",'\n',rf_conf_mat)
```

```
confusion matrix for lr_model
 [[6258  562]
 [ 851 1378]]
```

```
adb_conf_mat = confusion_matrix(y_test,adb_yprad)
print("confusion matrix for lr_model",'\n',adb_conf_mat)
```

```
confusion matrix for lr_model
 [[6377  443]
 [ 888 1341]]
```

```
svm_conf_mat = confusion_matrix(y_test,svm_yprad)
print("confusion matrix for svm_model",'\n',svm_conf_mat)
```

```
confusion matrix for svm_model
 [[6347  473]
 [ 988 1241]]
```

```
gdboost_conf_mat = confusion_matrix(y_test,gdboost_yprad)
print("confusion matrix for
gdboost_model",'\n',gdboost_conf_mat)
```

```
confusion matrix for gdboost_model
 [[6411  409]
 [ 880 1349]]
```

```
xgboost_conf_mat = confusion_matrix(y_test,xgboost_yprad)
print("confusion matrix for
xgboost_model",'\n',xgboost_conf_mat)
```

```
confusion matrix for xgboost_model
 [[6322  498]
 [ 745 1484]]
```

Checking classification report for each model:

```
lr_report = classification_report(y_test,lr_yprad)
print(" lr classification_report" ,'\n',lr_report)

knn_report = classification_report(y_test,knn_yprad)
print(" knn classification_report" ,'\n',knn_report)

dt_report = classification_report(y_test,dt_yprad)
print(" dt classification_report" ,'\n',dt_report)


rf_report = classification_report(y_test,rf_yprad)
print(" rf classification_report" ,'\n',rf_report)


adb_report = classification_report(y_test,adb_yprad)
print(" adb classification_report" ,'\n',adb_report)


svm_report = classification_report(y_test,svm_yprad)
print(" svm classification_report" ,'\n',svm_report)


gdboost_report = classification_report(y_test,gdboost_yprad)
print(" gdboost classification_report" ,'\n',gdboost_report)


xgboost_report = classification_report(y_test,xgboost_yprad)
print(" xgboost classification_report" ,'\n',xgboost_report)


lr classification_report
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      6820
           1       0.71      0.61      0.66      2229

    accuracy                           0.84      9049
   macro avg       0.80      0.76      0.78      9049
weighted avg       0.84      0.84      0.84      9049

 knn classification_report
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      6820
           1       0.63      0.58      0.60      2229
```

```
    accuracy                           0.81      9049
   macro avg       0.75      0.74      0.74      9049
weighted avg       0.81      0.81      0.81      9049

 dt classification_report
              precision    recall  f1-score   support

           0       0.87      0.86      0.87      6820
           1       0.60      0.62      0.61      2229

    accuracy                           0.80      9049
   macro avg       0.74      0.74      0.74      9049
weighted avg       0.81      0.80      0.81      9049


rf classification_report
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      6820
           1       0.71      0.62      0.66      2229

    accuracy                           0.84      9049
   macro avg       0.80      0.77      0.78      9049
weighted avg       0.84      0.84      0.84      9049

 adb classification_report
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      6820
           1       0.75      0.60      0.67      2229

    accuracy                           0.85      9049
   macro avg       0.81      0.77      0.79      9049
weighted avg       0.85      0.85      0.85      9049

 svm classification_report
              precision    recall  f1-score   support

           0       0.87      0.93      0.90      6820
           1       0.72      0.56      0.63      2229

    accuracy                           0.84      9049
   macro avg       0.79      0.74      0.76      9049
weighted avg       0.83      0.84      0.83      9049


gdboost classification_report
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      6820
           1       0.77      0.61      0.68      2229

    accuracy                           0.86      9049
   macro avg       0.82      0.77      0.79      9049
weighted avg       0.85      0.86      0.85      9049



 xgboost classification_report
              precision    recall  f1-score   support
```

```
              0       0.89      0.93      0.91      6820
              1       0.75      0.67      0.70      2229

    accuracy                            0.86      9049
   macro avg       0.82      0.80      0.81      9049
weighted avg       0.86      0.86      0.86      9049
```

As our data was imbalanced and in this case we have to consider F1-score and there are 2 models “xgboost ,‘gdboost”’ giving the scores above 90% and rest of below 90%. In order to check our model is overfitted or not we are checking the cross validation for 2 models which are giving the scores above 90%.

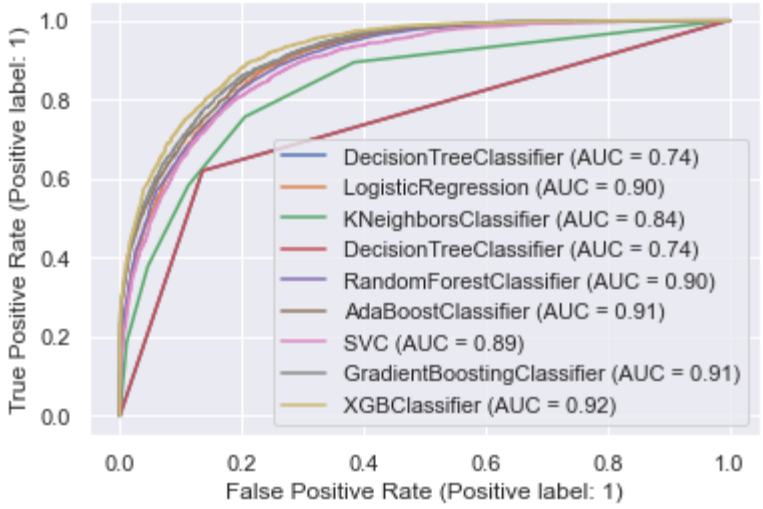
ROC AUC Curve:

Another way to evaluate and compare your binary classifier is provided by the ROC AUC Curve. This curve plots the true positive rate (also called recall) against the false positive rate (ratio of incorrectly classified negative instances), instead of plotting the precision versus the recall.

```
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import plot_roc_curve


#importing the ric and auc from sklearn and predect the
x_test and
checking the roc_auc_score


print(roc_auc_score(y_test,lr.predict(X_test)))
print(roc_auc_score(y_test,knn.predict(X_test)))
print(roc_auc_score(y_test,dt.predict(X_test)))
print(roc_auc_score(y_test,rf.predict(X_test)))
print(roc_auc_score(y_test,adb.predict(X_test)))
print(roc_auc_score(y_test,svm.predict(X_test)))
print(roc_auc_score(y_test,gdboost.predict(X_test)))
print(roc_auc_score(y_test,xgboost.predict(X_test)))
```

```
0.7649625241254642
0.7351546003165419
0.7419649212131736
0.767904876928886
0.7683295311470104
0.7436985339874673
0.772616726462296
0.7963744377303185
```

```
#lets find roc curve to check best fittted model


disp = plot_roc_curve(dt,X_test,y_test)
plot_roc_curve(lr,X_test,y_test,ax=disp.ax_)  # here ax_ for
axis with confustion matrics
plot_roc_curve(knn,X_test,y_test,ax=disp.ax_)
plot_roc_curve(dt,X_test,y_test,ax=disp.ax_)
plot_roc_curve(rf,X_test,y_test,ax=disp.ax_)
plot_roc_curve(adb,X_test,y_test,ax=disp.ax_)
plot_roc_curve(svm,X_test,y_test,ax=disp.ax_)
plot_roc_curve(gdboost,X_test,y_test,ax=disp.ax_)
plot_roc_curve(xgboost,X_test,y_test,ax=disp.ax_)
plt.legend(prop = {'size':11}, loc ='lower right')
```



As per above also we can see our XGB classifier and Gradient Boosting classifier giving the best scores.

K-Fold Cross Validation :

K-Fold Cross Validation randomly splits the training data into **K subsets called folds**. Let's image we would split our data into 3 folds (K = 3). Our random forest model would be trained and evaluated 4 times, using a different fold for evaluation every time.

Importing cross validation from sklearn.

```
from sklearn.model_selection import KFold,cross_val_score

k_f = KFold(n_splits=4,shuffle=True)
k_f
```

Now we have imported the Kfold cross validation , it will give 3 scores as we have gave parameter -n_split -3. Hence we are taking the mean of each model scores.

```
print("Mean of Cross validation score for gdboost
model","=>",cross_val_score(gdboost,X,y,cv=5).mean())

print("Cross validation score for xgboost
model","=>",cross_val_score(xgboost,X,y,cv=5).mean())

Cross validation score for gdboost  model =>
0.8676757809940992

Cross validation score for xgboost model =>
0.8670137849256147
```

As per above we can see our 'xgboost' model giving the less cross validation score. so we will consider this model is best for our prediction. Now will try hyperparameter tuning to check chances of accuracy increase.

HyperParameter Tuning !

Below you can see the code of the hyperparamter tuning for the parameters criterion, max_depth, subsample and learning_rate,random_state.

```
from sklearn.model_selection import GridSearchCV

xgboost.get_params().keys()  # to check the parameters

parm_grid  = {'max_depth' : [3,4],
              'subsample' : [0.5,0.8],
              'learning_rate': [0.1],
              'min_child_weight' : [1,2],
              'random_state' : [4,5]}

parm_grid

{'max_depth': [3, 4],
 'subsample': [0.5, 0.8],
 'learning_rate': [0.1],
 'min_child_weight': [1, 2],
 'random_state': [4, 5]}



#giving above parameters to our model and behalf of this
will train it again.

gridsearch = GridSearchCV(xgboost, param_grid = parm_grid ,
cv=5)

#fit the model using given paramters

gridsearch.fit(X_train,y_train)

#traning the model now
```

```
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0, gpu_id=-1,
                                     importance_type='gain',
                                     interaction_constraints='',
                                     learning_rate=0.300000012,
                                     max_delta_step=0, max_depth=6,
                                     min_child_weight=1, missing=nan,
                                     monotone_constraints='()',
                                     n_estimators=100, n_jobs=2,
                                     num_parallel_tree=1, random_state=0,
                                     reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, subsample=1,
                                     tree_method='exact', validate_parameters=1,
                                     verbosity=None),
             param_grid={'learning_rate': [0.1], 'max_depth': [3, 4],
                         'min_child_weight': [1, 2], 'random_state': [4, 5],
                         'subsample': [0.5, 0.8]})
```

```
gridsearch.best_params_
```

```
#printing the best parameters
```

```
{'learning_rate': 0.1,
 'max_depth': 4,
 'min_child_weight': 2,
 'random_state': 4,
 'subsample': 0.8}
```

```
xgboost_tuning=XGBClassifier(learning_rate=0.1,max_depth=4,m
in_child_weight=2,random_state=4,subsample=0.8)


#supplying best parameters to our model


xgboost_tuning.fit(X_train,y_train)


#train the model


XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1,
gamma=0, gpu_id=-1,
              importance_type='gain',
interaction_constraints='',
              learning_rate=0.1, max_delta_step=0,
max_depth=4,
              min_child_weight=2, missing=nan,
monotone_constraints='()',
              n_estimators=100, n_jobs=2,
num_parallel_tree=1, random_state=4,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
subsample=0.8,
              tree_method='exact', validate_parameters=1,


#predicting the values using test data


xgboost_tuning_yprad = xgboost_tuning.predict(X_test)



#printing the classification report


xgboost_report =
classification_report(y_test,xgboost_tuning_yprad)
print(" xgboost classification_report" ,'\n',xgboost_report)
```

```
xgboost classification_report
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      6820
           1       0.76      0.63      0.69      2229

    accuracy                           0.86      9049
   macro avg       0.83      0.78      0.80      9049
weighted avg       0.86      0.86      0.86      9049
```

As we can observed in the hyper parameter tuning also we are getting almost same scores.

Saving Model

```
import pickle

Census_model = 'Census_model.pickle'

pickle.dump(xgboost_tuning,open(Census_model,'wb'))
```

Concluding Remarks

We started the our project to import various libraries and imported the dataset from GitHub. Observing the many important points like problem type and how many columns contains int ,float and object values. As per statistic observations we found huge variations among the features and we have used standard scaler to scale the variables. Besides this, we have identified there are 2 columns where instead the NaN it is in “?” so we deleted such rows. During this process we used seaborn and matplotlib to do the visualizations and converted categorical features into numeric using label encoder and pandas get_dummies function. Afterwards we started training different different machine learning models, picked one of them (Xgboost) and applied cross validation on it and we tried to tune model using hyperparameter tuning.

To conclude, There are many other ways also to improve the model accuracy like doing a more extensive feature engineering, by comparing and plotting the features against each other and identifying and removing the noisy features, Along with resampling the data in

case of imbalance or more extensive hyperparameter tuning on several machine learning models