In [ ]:

```
#*****************************************************************************
#
#               Problem : To predict if a person makes >50K using XGBoost
#
#
#*****************************************************************************
# Step 0: To Load the libraries, packages being used

import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import roc_auc_score,confusion_matrix
from sklearn.model_selection import KFold,cross_val_score
import matplotlib.pyplot as plt
%matplotlib inline


# To load the Data
train_set = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data", header = None)

test_set = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.test", skiprows = 1, he

col_labels = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status', 'occupation','relationshi
train_set.columns = col_labels
test_set.columns = col_labels

print(" The Training  Data Set are as : \n",'-'*80)

print(train_set.head(2))
print('-'*80)

print(" The  Test Data Set are as : \n",'-'*80)

print(test_set.head(2))

print( "\n The number of rows in training and test set \n",'-'*80)
print(train_set.shape)
print(test_set.shape)
```

```
train_rowcnt = train_set.shape[0]
test_rowcnt = test_set.shape[0]
train_percnt = (train_rowcnt/(train_rowcnt+test_rowcnt))

print( " The Training percentage of rows is : ",round(train_percnt,2)*100)
```

In [ ]:

```
# Step1 : Analysing the train Data by seeing info, describe, shape , nulls functions

print(" To combine the Train and Test DataSets to analyse together the Data ")
df = pd.concat([train_set,test_set], ignore_index= True)
print(df.shape)

print('-'*80)
print(df.head())

print(" The Total numbr of rows and columns in the  DataSet \n",'-'*80)
print(df.shape)

print(" Analyse  the Data by using info, describe and check for nulls, categorical fields ")
print('-'*80)

print(df.info())
print('-'*80)

print(df.describe())
print('-'*80)

print(df.isnull().sum())
print('-'*80)
```

In [ ]:

```
# Step2: Preprocessing the Data

# We observe , there are no nulls but lot of categorical fields which need to made numeric
#['workclass','education', 'marital_status', 'occupation','relationship', 'race', 'sex','native_country', 'wage_class

# using LabelEncoder to convert the Categorical fields to numeric in both train and test set
print(" The Categorical Fields  and their unique values \n",'-'*80)
print(df.workclass.unique())
print(df.education.unique())
print(df.marital_status.unique())

print(df.occupation.unique())
print(df.relationship.unique())
print(df.race.unique())
print(df.sex.unique())
print(df.native_country.unique())
print(df.wage_class.unique())

# Observing the Data , we see there are spaces as well as '?' character in some fields , using function
# trimAllColumns to remove spaces

def trimAllColumns(df):
    trimStrings = lambda x: x.strip() if type(x) is str else x
    return df.applymap(trimStrings)

df = trimAllColumns(df)
df = df.replace('?',np.nan)

# also in the wage_Class we need only see if greater than 50K or not , so correctly some rows
df = df.replace('>50K.','>50K')
df = df.replace('<=50K.','<=50K')


```

In [ ]:
```
# since the number of nulls is very less as compared to size of the total rows , so we can ignore the nulls
# or drop it

df =  df.dropna()

```

In [ ]:
```
# Changing the Categorical Values to  numeric values

le = LabelEncoder()

cat_features = ['workclass','education','marital_status','occupation','relationship','race','sex','native_country','w


for att in cat_features:
    df[att] = le.fit_transform(df[att].astype(str))

print(" The Categorical fields changed to numerical by Labelencoder \n",'-'*80)
print(df.head())

```

In [ ]:
```
# Step3 : Visualisation of train Data to see relationship with Wage_class attribute
# here instead of taking all the rows , a sample size of 10000 records is used


features = df.columns
y = 'wage_class'

print(" Visualising the Data to see the relationship with wage_class")
column_cnt = len(features)
i = 0
while i< column_cnt:
    sns.pairplot(df.sample(10000),x_vars=features[i:i+4],y_vars=y,kind="reg")
    i = i+4

```

In [ ]:
```
# The Full DataSets

X = df
X = X.drop('wage_class',axis=1)
y = df['wage_class']
y = y.values.reshape(-1,1)
```

In [ ]:
```
#! pip install xgboost
#Step4: Splitting the combine Dataset after preprocessing based on the previous training set size

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=(1-train_percnt), random_state=2)

# X_train = df[:train_set.shape[0]]
# X_test = df[train_set.shape[0]:]
print(" The number of rows amd columns in the training and test set now are :\n")
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)

print(" The training set : \n",'-'*80)
print(X_train.head(2))

print(" The test set : \n",'-'*80)
print(X_test.head(2))

```

In [ ]:

```
# Step4: XgBoost model creation from sklearn library
from sklearn.metrics import accuracy_score
from xgboost.sklearn import XGBClassifier
from sklearn.model_selection import KFold,cross_val_score
import warnings


# to remove the warnings coming in the output
if __name__ == '__main__':
    warnings.filterwarnings(action='ignore', category=DeprecationWarning)


# setting the parameters
params = {
    'objective': 'binary:logistic',
    'max_depth': 3,
    'learning_rate': 0.1,
    'silent': 1,
    'n_estimators': 26,
    'eval_metric':['auc','logloss']

}

# Seeing the Performance For Xgboost Classifier with customised paramters
# Sklearn Xgboost Classifier 1

bst = XGBClassifier(**params)

y_train = y_train.ravel()

# training the Classfier with the training set
bst.fit(X_train, y_train)
#, eval_set=[(X_train, y_train), (X_test, y_test)]

print(" Xgboost Classifier 1 with Customised Parameters \n",'-'*80)

print(bst)

# predicting the labels for the test set
preds = bst.predict(X_test)

```

```
preds = pd.Series(preds)
print(preds.shape)
print(y_test.shape)

# y_test = y_test.values.reshape(-1,1)
preds = preds.values.reshape(-1,1)

correct=0
for i in range(len(preds)):
    if (y_test[i] == preds[i]):
        correct += 1
if y_test.size >0 :
    acc = accuracy_score(y_test, preds)

print('Predicted correctly: {0}/{1}'.format(correct, len(preds)))
print('Accuracy calculated  manually: ', correct*100/len(preds), '%')
print('Accuracy calculated by accuracy score function: ', acc*100, '%')
print('Error: {0:.4f}'.format((1-acc)*100))
print('-'*80)

print(X.shape)
print(y.shape)

y = y.ravel()
if y.size > 0 :
    scores = cross_val_score(bst,X,y, scoring = "accuracy",cv=15)
    print(" The accuracy by cross validation function is :",scores.mean())

```

In [ ]:

```
# Seeing the Performance for Xgboost Classifier with default set of paramters
# Classfier with DEfault parameters

bst1 = XGBClassifier()

# training the Classfier with training set
#st1.fit(X_train,y_train ,eval_set=[(X_train, y_train), (X_test, y_test)])

bst1.fit(X_train,y_train)

print(" Xgboost Classifier 2 with Default parameters \n",'-'*80)

print(bst1)

# predicting the labels for the testing set
preds1 = bst1.predict(X_test)

preds1 = pd.Series(preds1)

print(y_test.shape)
print(preds1.shape)

preds1 = preds1.values.reshape(-1,1)

acc_def = round(accuracy_score(y_test, preds1),2)

print('Accuracy for default classifier by accuracy score function: ', acc_def*100, '%')

scores = cross_val_score(bst1,X,y,scoring ="accuracy",cv=15)
print(" The accuracy by cross validation function is :",round(scores.mean(),2)*100)

```

In [ ]:

```
# Predicting the Performance by using directly Xgboost Classifier
print(y_train.shape)

# converting the training  and test set to sparse matrix by using DMatrix
XG_train = xgb.DMatrix(X_train,y_train)
XG_test = xgb.DMatrix(X_test,y_test)


params = {
    'objective':'binary:logistic',
    'max_depth':4,
    'silent':1,
    'eta':1
    }

num_rounds = 26
watchlist  = [(XG_test,'test'), (XG_train,'train')]

bst2 = xgb.train(params, XG_train, num_rounds,watchlist)
#bst = xgb.train(params, XG_train, num_rounds)

print(bst2)

preds_prob = bst2.predict(XG_test)

print(preds_prob)

print("\nTrain possible labels: ")
print(np.unique(XG_train.get_label()))

print("\nTest possible labels: ")
print(np.unique(XG_test.get_label()))

labels = XG_test.get_label()
preds = preds_prob > 0.5 # threshold

correct = 0

for i in range(len(preds)):
    if (labels[i] == preds[i]):
        correct += 1
```

```
42
43 acc = accuracy_score(labels,preds)
44
45 print('\n Predicted correctly: {0}/{1}'.format(correct, len(preds)))
46 print('Accuracy calculated manually : ', correct*100/len(preds), '%')
47 print('Accuracy by accuracy score function: ', acc*100, '%')
48 print('Error: {0:.4f}'.format(1-correct/len(preds)))
49
50
51 print(" The ROC score is : %.3f "%roc_auc_score(labels,preds))
52 print(" The confusion matrix is :\n")
53 print(confusion_matrix(labels,preds))
54
```

In [ ]:

```
1 # finding the Kfold and Cross Val score for this algoritm
2
3 from sklearn.model_selection import KFold,cross_val_score
4 cv_results = xgb.cv(params, XG_train, num_rounds,nfold=5,metrics="auc")
5 print(cv_results.head())
6 print('-'*80)
7 print(cv_results.tail(1))
8 print(cv_results.mean())
9
```