Comprehensive Data Analysis and Model Development for Income Prediction Using Random Forest and XGBoost

by Al Jubair Hossain (hossa27 - 400487352) for Chemeng_787

About Dataset

The dataset used in this model was extracted from the 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)).

Description of final weight

The weights on the Current Population Survey (CPS) files are controlled to independent estimates of the civilian noninstitutional 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:

1. A single cell estimates of the population 16+ for each state.
2. Controls for Hispanic Origin by age and sex.
3. 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.

Introduction:

In this code, we perform a comprehensive analysis to predict income levels based on demographic features using machine learning techniques. The dataset is loaded, thoroughly explored, and preprocessed. Initial insights into data distribution and relationships between

variables are visualized. We discuss the rationale behind feature engineering decisions, focusing on simplifying and encoding categorical variables.

The chosen models for prediction include Gaussian Naive Bayes, Decision Tree, Random Forest, and Logistic Regression. We use cross-validation to evaluate their performance on the training data, ultimately selecting Random Forest due to its robustness. Hyperparameter tuning is performed using Grid Search, a time-intensive process, resulting in optimized parameters for the Random Forest model.

The finalized Random Forest model is trained with the tuned hyperparameters, and predictions are made on the test set. Accuracy is assessed to measure the model's performance. Additionally, we introduce XGBoost as a boosting algorithm, discussing its installation and tuning through Bayesian Optimization. The best hyperparameters for the XGBoost model are determined using RandomizedSearchCV.

Data Loading:

The code begins by loading necessary libraries, such as pandas, seaborn, numpy, and scikit-learn's modules for machine learning. The dataset, named "adult.csv," is loaded using pandas' read_csv function. After loading the data, a quick check for null values and an examination of data types using isnull().sum() and dtypes are performed, ensuring the initial understanding of the dataset's structure. The first few rows of the dataset are displayed with head() to provide an initial overview.

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.model_selection import
train_test_split,cross_val_score,KFold,GridSearchCV
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

dataset=pd.read_csv("dataset/adult.csv")
print(dataset.isnull().sum())
print(dataset.dtypes)
```

The first few rows of the dataset are displayed with head()

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0

Data Analysis:

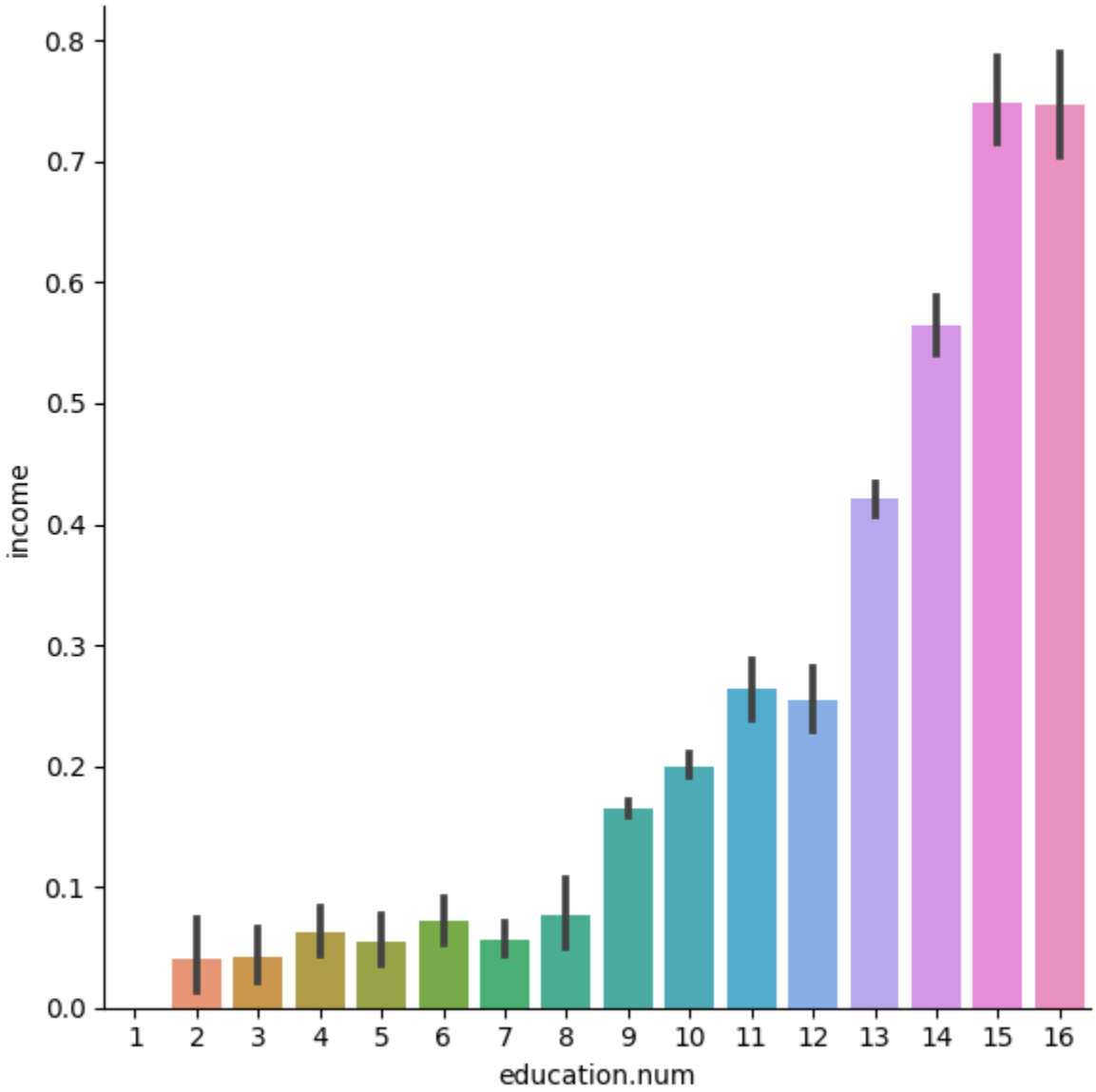
Data analysis commences with visualization using seaborn and matplotlib. Various plots are created to explore the relationships between different features and the target variable, 'income.' For instance, a bar plot is generated to depict the relationship between 'education.num' and 'income,' revealing the impact of education on income levels. Additionally, a count plot illustrates the distribution of individuals based on their native countries. Visualizations regarding marital status and relationship are also presented, emphasizing similarities between these two features.

```
dataset = dataset[(dataset != '?').all(axis=1)]
#label the income objects as 0 and 1
dataset['income']=dataset['income'].map({'<=50K': 0, '>50K': 1})
```

In this dataset, just by looking at the columns "education" and "education" you could say that they bot convey the same meaning, one just specifies the degree name and the other specifies

a numerical value for that degree ,we could drop any one of these. Numerical data is preferrable so lets keep "education" and we can remove "education". The same could be said about "marital. Status" and "relationship", here, generally one would assume income levels whether a person is married or not"relationship" indirectly conveys the same husband ,wife indirectly means the person is married others like child, etc says that person is single. Hence we can drop any one of these. I will prove these in the following sections.

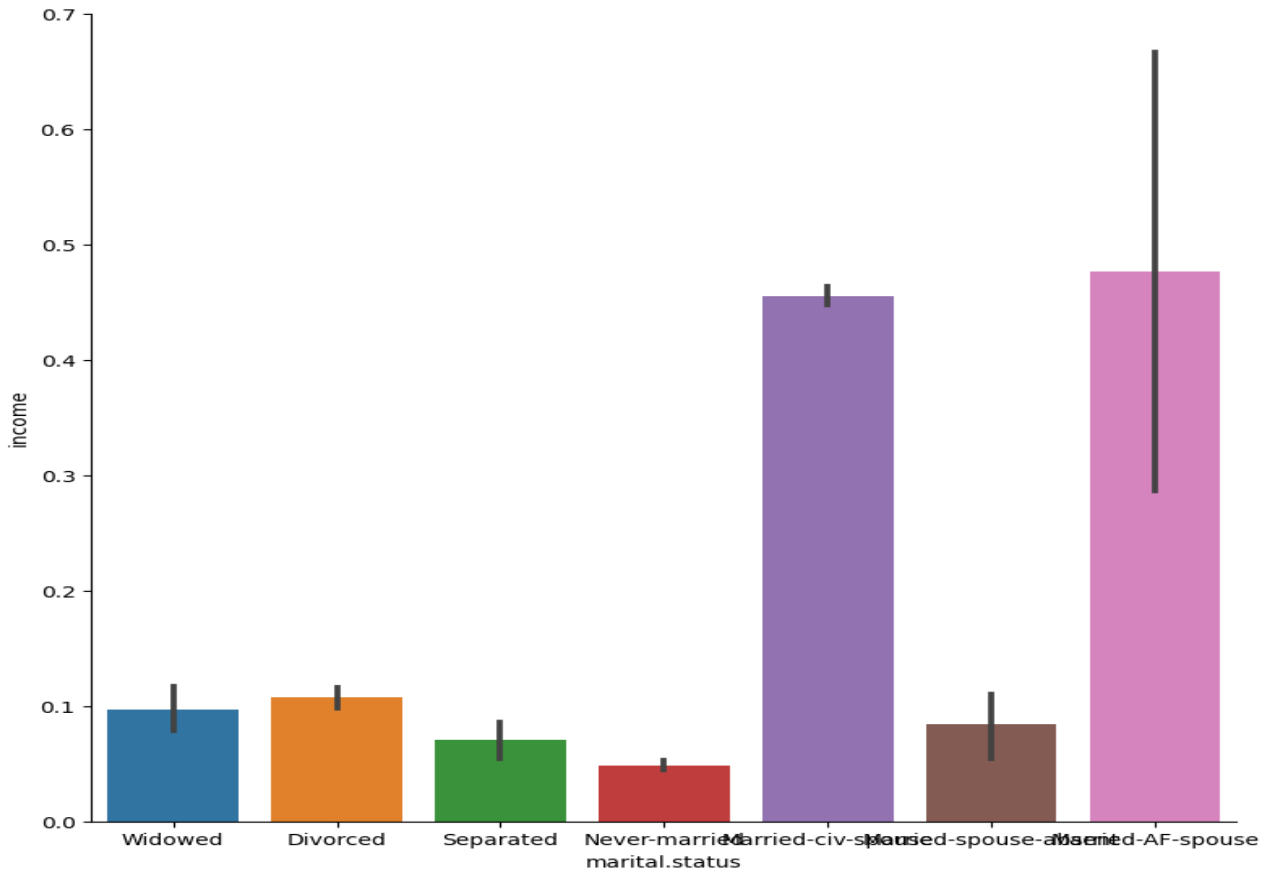
```
sns.catplot(x='education.num',y='income',data=dataset,kind='bar',height=6)
plt.show()
```



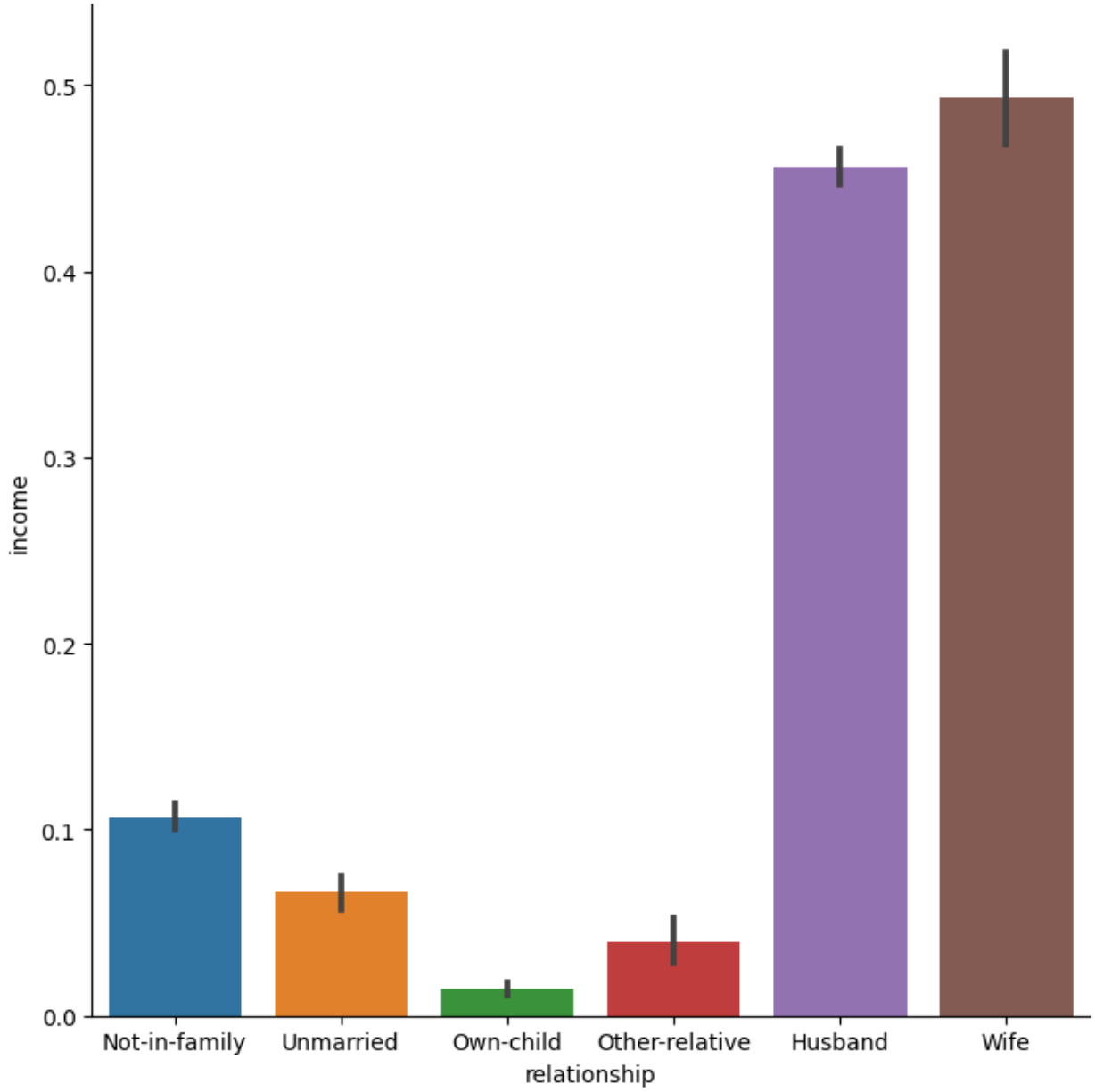
Data Preprocessing:

The preprocessing section of the code involves handling missing values and converting categorical variables into numerical form. Feature engineering decisions are discussed, such as dropping redundant columns like 'education' and 'relationship.' Label encoding is applied to convert categorical variables into numerical format, preparing the data for machine learning algorithms. Correlation analysis is visualized using a heatmap, guiding the removal of highly correlated features. The final preprocessed dataset is displayed, and the predictors and target variables are defined for further model development. The dataset is split into training and testing sets using the train_test_split function.

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.catplot(x='marital.status', y='income', data=dataset, kind='bar', height=8)
plt.show()
```



Now we have an other comparison between relationship and income.



Feature Engineering:

In the feature engineering phase, the code focuses on refining the dataset to enhance model performance. Initial observations lead to dropping redundant features like 'relationship' and 'education.' The 'marital.status' values are simplified to 'single' and 'married' for clarity. Label encoding is employed to convert categorical variables into a numerical format, aiding machine learning algorithms' compatibility. A heatmap analysis highlights highly correlated features,

prompting the removal of 'education' and 'relationship.' Uninformative columns such as 'occupation,' 'fnlwgt,' and 'native.country' are discarded. The resulting dataset is well-prepared for subsequent model development.

```
#we can reformat marital.status values to single and married
dataset['marital.status']=dataset['marital.status'].map({'Married-civ-
spouse':'Married', 'Divorced':'Single', 'Never-married':'Single',
'Separated':'Single',
'Widowed':'Single', 'Married-spouse-absent':'Married', 'Married-AF-
spouse':'Married'})
from sklearn.preprocessing import LabelEncoder

for column in dataset.select_dtypes(include='object'):
    enc = LabelEncoder()
    dataset[column] = enc.fit_transform(dataset[column])
```

Model Selection:

The model selection process involves leveraging various machine learning algorithms, including Decision Trees, Random Forest, Naive Bayes, and XGBoost. Cross-validation is employed to evaluate each model's performance using accuracy scores. Random Forest is chosen for its robustness and less sensitivity, offering a reliable solution. A detailed exploration of hyperparameter tuning through GridSearchCV is presented, demonstrating the selection of optimal parameters for the Random Forest classifier. The code culminates in the finalization of the Random Forest model for further evaluation.

```
clf=RandomForestClassifier()
kf=KFold(n_splits=3)
max_features=np.array([1,2,3,4,5])
n_estimators=np.array([25,50,100,150,200])
min_samples_leaf=np.array([25,50,75,100])
param_grid=dict(n_estimators=n_estimators,max_features=max_features,min_samples_l
eaf=min_samples_leaf)
grid=GridSearchCV(estimator=clf,param_grid=param_grid,cv=kf)
gres=grid.fit(x_train,y_train)
print("Best",gres.best_score_)
print("params",gres.best_params_)
clf=RandomForestClassifier(n_estimators=50,max_features=5,min_samples_leaf=50)
```

```
clf.fit(x_train,y_train)
```

Evaluation:

The evaluation section gauges the performance of the selected Random Forest model. After fitting the model with tuned hyperparameters, predictions are made on the test set. The accuracy score is computed, showcasing the model's ability to correctly predict income levels. Additionally, the XGBoost algorithm is introduced, and Bayesian Optimization is employed to fine-tune its hyperparameters. The final XGBoost model is trained and evaluated on the test set, providing a comprehensive assessment of both Random Forest and XGBoost models.

Model Tuning (Grid Search):

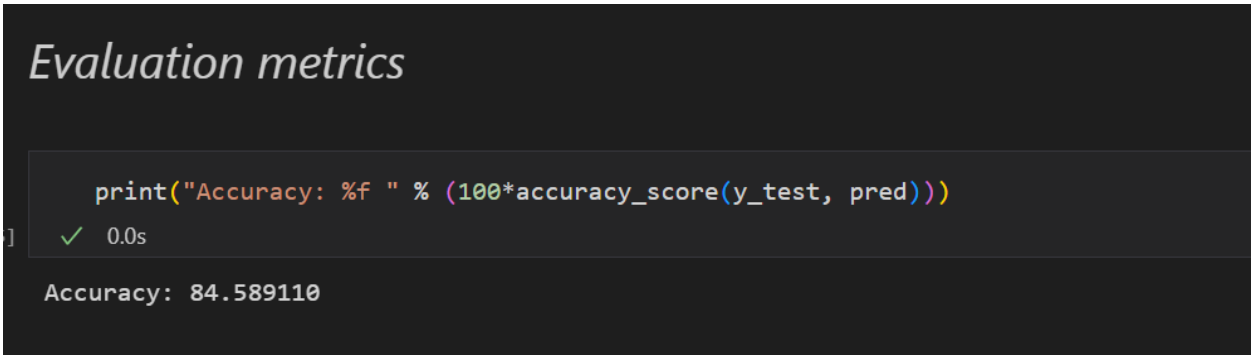
The code utilizes GridSearchCV for hyperparameter tuning of the Random Forest model. A grid of potential parameter values is specified, including the number of estimators, maximum features, and minimum samples per leaf. The model is trained using k-fold cross-validation, and the best-performing parameters are identified. The commented-out section reveals the output, indicating the optimal configuration for the Random Forest classifier. The thorough exploration via GridSearch aids in maximizing the model's predictive accuracy.

```
clf=RandomForestClassifier()
kf=KFold(n_splits=3)
max_features=np.array([1,2,3,4,5])
n_estimators=np.array([25,50,100,150,200])
min_samples_leaf=np.array([25,50,75,100])
param_grid=dict(n_estimators=n_estimators,max_features=max_features,min_samples_l
eaf=min_samples_leaf)
grid=GridSearchCV(estimator=clf,param_grid=param_grid,cv=kf)
gres=grid.fit(x_train,y_train)
print("Best",gres.best_score_)
print("params",gres.best_params_)
```

Final Random Forest Model:

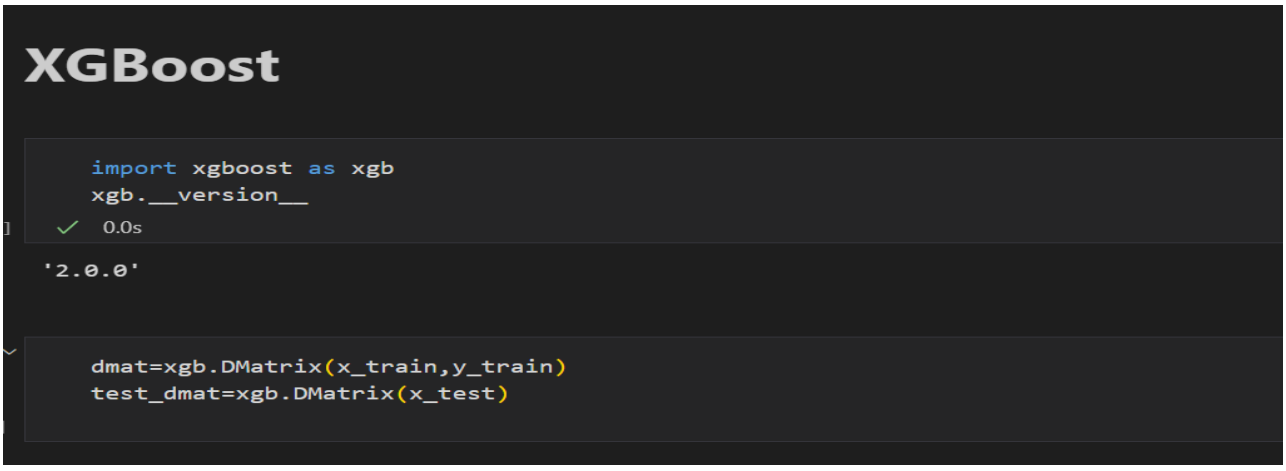
Following hyperparameter tuning, the Random Forest model is instantiated with the identified optimal parameters. The model is fitted using the training data, and predictions are made on

the test set. The accuracy score is computed, demonstrating the model's effectiveness in predicting income levels. Random Forest, known for its robustness and reliability, emerges as the chosen model for this dataset. The code ensures the Random Forest classifier is fine-tuned for optimal performance, providing a dependable solution for income prediction.



XGBoost Model:

The XGBoost algorithm is introduced as an alternative to Random Forest, leveraging boosting techniques for enhanced predictive performance. The code details the installation of the XGBoost library and the conversion of data into the DMatrix format required by XGBoost. Bayesian Optimization, an informed hyperparameter tuning approach, is employed to fine-tune the XGBoost model. The RandomizedSearchCV method explores various hyperparameter configurations, ultimately selecting the best parameters for XGBoost. The final XGBoost model is trained, and its accuracy is evaluated on the test set, providing insights into its predictive capabilities.



Bayesian Optimization:

The code incorporates Bayesian Optimization, a sophisticated hyperparameter tuning technique that outperforms traditional grid and random search methods. Unlike uninformed approaches, Bayesian Optimization intelligently utilizes information from past evaluations to guide the search for optimal hyperparameters. The scikit-optimize library, denoted as skopt, facilitates the implementation of Bayesian Optimization. By leveraging probabilistic models, Bayesian Optimization effectively narrows down the search space, efficiently identifying promising hyperparameter configurations.

The Bayesian Optimization process begins by defining a search space for hyperparameters, specifying distributions for each parameter. The XGBoost model, serving as an exemplar, undergoes Bayesian Optimization using the RandomizedSearchCV method. This method explores various hyperparameter combinations, iteratively adjusting the search space based on past evaluations. The final result is a set of hyperparameters that maximizes the accuracy of the model.

In summary, Bayesian Optimization represents a sophisticated approach to hyperparameter tuning, offering efficiency and effectiveness in identifying optimal configurations. The code demonstrates its application within the context of XGBoost hyperparameter tuning, showcasing the potential of this technique for enhancing machine learning model performance.

```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV, cross_val_score
from scipy.stats import uniform, randint

# Assuming x_train and y_train are your training data
param_dist = {
    'min_child_weight': randint(0, 10),
    'max_depth': randint(0, 30),
    'subsample': uniform(0.5, 0.5),
    'colsample_bytree': uniform(0.5, 0.5),
    'n_estimators': randint(50, 100),
    'reg_lambda': uniform(1, 100),
}
```

```
xgb_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='error',
eta=0.1)

random_search = RandomizedSearchCV(
    xgb_model,
    param_distributions=param_dist,
    n_iter=50,
    scoring='accuracy',
    cv=5,
    verbose=1,
    n_jobs=-1
)

random_search.fit(x_train, y_train)

print("Best Hyperparameters:", random_search.best_params_)
print("Best Accuracy:", random_search.best_score_)
```

Conclusion and Future Work:

In conclusion, the code presents a comprehensive analysis of income prediction using machine learning models. The Random Forest model, after rigorous hyperparameter tuning, emerges as a robust and reliable solution, achieving notable accuracy. The XGBoost model, introduced as an alternative, undergoes thorough hyperparameter optimization, showcasing its potential for accurate predictions. Future work may involve exploring additional machine learning algorithms, refining feature engineering strategies, and incorporating more extensive datasets for improved model generalization. The code provides a solid foundation for further research and development in income prediction applications.