

**School of Electrical Engineering and Computer Science (SEECS)**

**CS-471 Machine Learning**

**Class: BESE-12B**

**Lab 11: Open-Ended Lab Report**

**Instructor: Dr. Muhammad Daud Abdullah Asif**

**Lab Engineer: Ms. Irum Tariq Bhatti**

**Submitted by Jamal Ahmad Khan**

**Table of Contents**

[**1.** **Introduction** 3](#_Toc166813923)

[**1.1 Background Information** 3](#_Toc166813924)

[**1.2 Objective of the Lab** 3](#_Toc166813925)

[**2.** **Dataset Description** 4](#_Toc166813926)

[**2.1 Overview of the UCI Adult Income Dataset** 4](#_Toc166813927)

[**2.2 Features and Target Variable** 4](#_Toc166813928)

[**3.** **Importing the Libraries** 4](#_Toc166813929)

[**4.** **Importing the Dataset** 6](#_Toc166813930)

[**5.** **Data Preprocessing** 6](#_Toc166813931)

[**5.1 Shape of Dataset** 6](#_Toc166813932)

[**5.2 Head() method** 7](#_Toc166813933)

[**5.3 Correcting unique values in Target column** 7](#_Toc166813934)

[**5.4 Handling Missing Values** 7](#_Toc166813935)

[**5.5 Encoding Categorical Columns** 9](#_Toc166813936)

[**5.6 Dropping Low Correlated Columns** 9](#_Toc166813937)

[**5.7 Splits the Dataset into Training set and Test set** 10](#_Toc166813938)

[**5.8 Feature Scaling** 11](#_Toc166813939)

[**6.** **Approach used for Applying ML models** 11](#_Toc166813940)

[**7.** **Results and Comparison** 11](#_Toc166813941)

[**7.1 Logistic Regression** 12](#_Toc166813942)

[**7.2 KNN (n = 4)** 12](#_Toc166813943)

[**7.3 SVM (kernel = rbf)** 13](#_Toc166813944)

[**7.4 Naïve Bayes** 13](#_Toc166813945)

[**7.5 Decision Tree** 14](#_Toc166813946)

[**7.6 Random Forest** 14](#_Toc166813947)

[**8.** **Conclusion** 14](#_Toc166813948)

[**9.** **Recommendations for Future Work** 15](#_Toc166813949)

**OEL Report**

# **Introduction**

Predicting how much money someone makes is really important for things like deciding government policies, figuring out how to advertise products, and studying society. Using machine learning to guess whether someone earns more than $50,000 a year has become a big deal in data science. This lab is all about looking closely at a dataset called UCI Adult Income Dataset to build a strong machine learning model. The goal is to make this model really good at sorting people into different income groups based on information about them.

## **1.1 Background Information**

The UCI Adult Income Dataset comes from a big collection of information called the UCI Machine Learning Repository. It includes lots of details about people's lives, like how old they are, their education, job, and whether they're married. The main aim of this dataset is to figure out if someone makes more than $50,000 each year, based on all these details about them. This kind of prediction is really important for understanding why some people earn more than others and how we can help make things fairer for everyone.

## **1.2 Objective of the Lab**

The objective of this lab is to define a machine learning model that predicts in accurately classifying individuals into income categories based on their demographic and socioeconomic features. In this lab, we aim to:

* Conduct exploratory data analysis (EDA) to gain insights about the dataset.
* Preprocess the data by handling missing values, encoding categorical variables, and scaling numerical features.
* Select appropriate machine learning algorithms, including logistic regression, k-nearest neighbors (KNN), and support vector machines (SVM), naive Bayes, decision trees, and random forests.
* Implement each selected algorithm, fine-tuning hyper parametersd where necessary, and evaluate their performance using cross-validation techniques.
* Compare the performance of different models and analyze the factors contributing to their effectiveness in predicting income levels.

Through these steps, we aim to develop a robust machine learning model with the highest performance scores, providing valuable insights into the socioeconomic factors influencing income levels and contributing to the broader understanding of income prediction in data science.

# **Dataset Description**

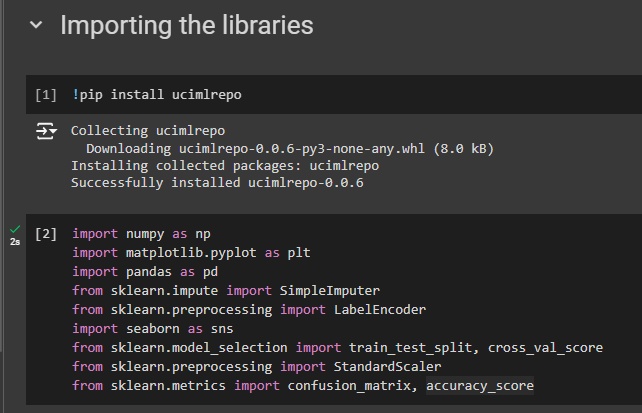
## **2.1 Overview of the UCI Adult Income Dataset**

The UCI Adult Income Dataset is a collection of information about people's lives, gathered from various sources. It contains details like age, education, job, and marital status. This dataset is commonly used in machine learning to predict whether someone earns more than $50,000 a year based on these details.

## **2.2 Features and Target Variable**

The dataset includes several features, which are characteristics or attributes of individuals. Some of the features include age, education level, type of job, and marital status. The target variable is the attribute we want to predict, which in this case is whether someone earns more than $50,000 a year. In machine learning terms, the features are used to make predictions about the target variable.

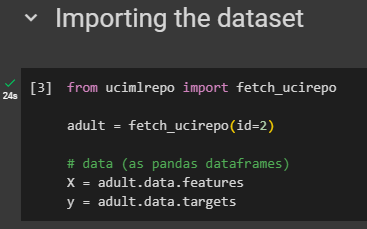
# **Importing the Libraries**



Following is a short description for each of these libraries that we have imported:

* **ucimlrepo**: This is a Python package that provides an easy interface to access datasets from the UCI Machine Learning Repository. It allows users to download and load datasets directly into their Python environment for analysis and modeling.
* **numpy (np)**: NumPy is a powerful library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.
* **matplotlib.pyplot (plt)**: Matplotlib is a popular plotting library in Python used for creating static, interactive, and animated visualizations. The pyplot module provides a MATLAB-like interface for creating plots and visualizations.
* **pandas (pd)**: Pandas is a versatile data manipulation library in Python. It provides data structures like DataFrame and Series, along with a wide range of functions for data manipulation, exploration, and analysis.
* **sklearn.impute.SimpleImputer**: This module from the scikit-learn library provides a simple and flexible strategy for imputing missing values in datasets. It allows users to replace missing values with statistical measures like mean, median, or mode along specified axes.
* **sklearn.preprocessing.LabelEncoder**: LabelEncoder is a utility class in scikit-learn used for encoding categorical labels as integer numbers. It converts categorical variables into numerical representations, which can be more easily understood by machine learning algorithms.
* **seaborn (sns)**: Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics. Seaborn is particularly useful for creating complex visualizations like heatmaps and pair plots.
* **sklearn.model\_selection.train\_test\_split**: This function from scikit-learn is used to split datasets into random train and test subsets. It is commonly used in machine learning to evaluate model performance on unseen data.
* s**klearn.preprocessing.StandardScaler**: StandardScaler is a preprocessing module in scikit-learn used for standardizing features by removing the mean and scaling to unit variance. It transforms data to have a mean of 0 and a standard deviation of 1, which can improve the performance of certain machine learning algorithms.
* **sklearn.metrics.confusion\_matrix**: Confusion matrix is a performance measurement tool for classification models. It presents a summary of prediction results, showing the number of true positives, true negatives, false positives, and false negatives.
* **sklearn.metrics.accuracy\_score**: Accuracy score is a metric used to evaluate the performance of classification models. It calculates the proportion of correctly classified instances out of all instances in the dataset.

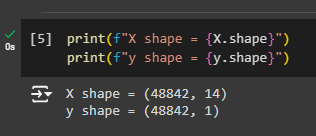
# **Importing the Dataset**



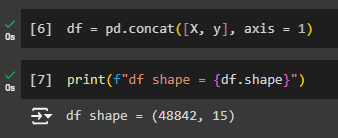
The code given in the snippet utilizes the fetch\_ucirepo function from the ucimlrepo package to retrieve a dataset from the UCI Machine Learning Repository. In this specific case, it fetches the dataset with the ID 2, which is commonly known as the "Adult" dataset.

# **Data Preprocessing**

## **5.1 Shape of Dataset**

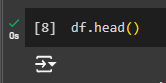


First of all in the data preprocessing stage we checked the dimension of both X and y where we have stored the adult data features and target values.



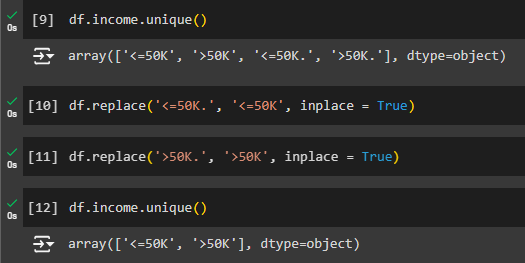
After, that we created a data frame by using the concat function from pandas library to combine the two X and y (features and target values) with each other so that we can further do our data preprocessing easily.

## **5.2 Head() method**



Then we check the dataset first five rows using head function to gain an overview of data.

## **5.3 Correcting unique values in Target column**



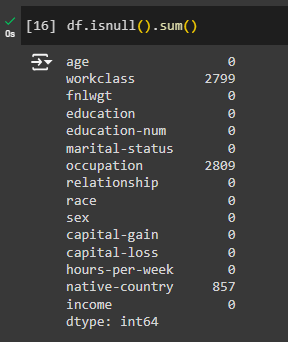
Initially, some labels end with a dot (like '<=50K.'), while others don't. This can cause confusion. So, this code replaces the labels with dots (like '<=50K.') with labels without dots (like '<=50K'). This makes all the labels consistent. Now, when we analyze or build models with the data, it's easier to understand and more accurate. Overall, this step ensures that our machine learning models can predict income levels better.

## **5.4 Handling Missing Values**

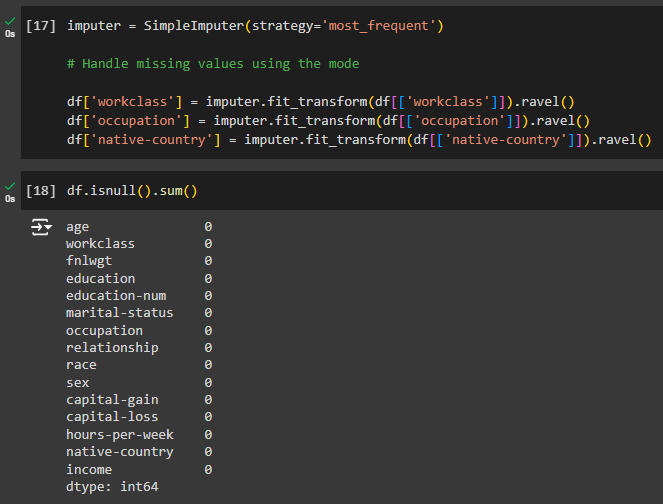
Handling missing values is crucial in data analysis and modeling. Missing information is common in real-world data and can impact the accuracy of our analysis. Strategies for handling missing values include removing them, filling them with a specific value, or using imputation techniques. Each approach has its advantages and disadvantages, depending on the data and analysis. Careful handling of missing values ensures more accurate results and better-informed decisions.



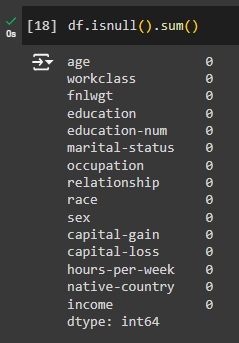
First, we replace ‘?’ values if any in our dataset with np.nan values because the machine learning techniques actually do not recognize ‘?’ values as empty values. So, if there are any these kind of values in our dataset, we replace them with np.nan values.



We checked the missing values in our dataset, then we calculate the total number of missing values in each column of the DataFrame using the isnull().sum() method. The result is a Series showing the count of missing values for each column. This step helps identify which columns contain missing data and informs the subsequent handling of these missing values.



We then fixed the missing information in the 'workclass', 'occupation', and 'native-country' columns of our DataFrame, df. We use a tool called SimpleImputer from scikit-learn, setting it up to replace missing values with the most common value in each column. Then, we apply this tool to each selected column. It figures out the most common value and fills in any missing spots with it. After that, we flatten the result to fit back into our DataFrame. This way, our dataset becomes more complete, making it easier to understand and analyze.

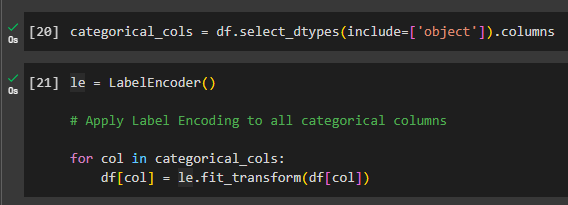


After handling the missing values, we again checked the missing information across our dataset and this time we can see that the dataset is free of any further missing data.

## **5.5 Encoding Categorical Columns**

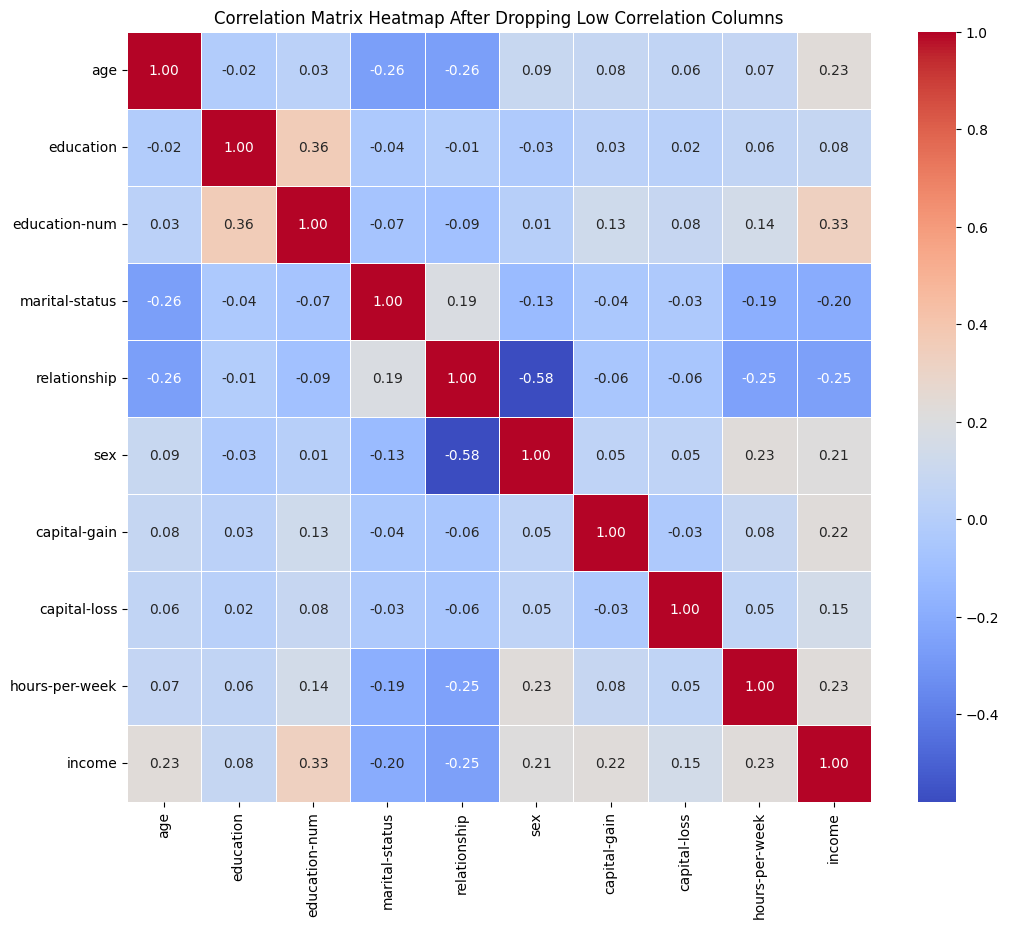
Encoding categorical columns involves converting categorical data into numerical values so that machine learning algorithms can process them effectively. This is done using techniques like Label Encoding, assigning unique numerical values to each category, and One-Hot Encoding, which creates binary columns for each category. By encoding categorical data, we enable algorithms to understand and analyze the categorical information, facilitating the creation of accurate machine learning models. This step is crucial for dealing categorical features in the dataset to make predictions and uncover patterns.

We have applied LabelEncoder for encoding the categorical columns.



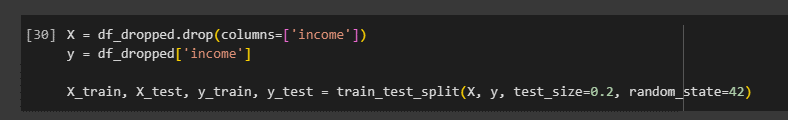
## **5.6 Dropping Low Correlated Columns**

The columns 'workclass', 'fnlwgt', 'occupation', 'race', 'native-country', and 'education' have been dropped from our dataset. This decision was made based on their low correlation with the target column, as observed through correlation values calculated using the corr() method or by visualizing the heatmap. By removing these less correlated columns, we streamline our dataset to focus on the most influential features for predicting the target variable, thus potentially improving the performance of our machine learning models.



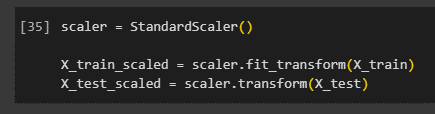
## **5.7 Splits the Dataset into Training set and Test set**

We split our preprocessed dataset into training and testing sets for model evaluation. The features are stored in the DataFrame X, excluding the target variable 'income', which is stored separately in the Series y. We then utilize the train\_test\_split function from scikit-learn to divide the data into training and testing subsets, with a test size of 20% of the total dataset. Setting the random\_state parameter ensures reproducibility of the split, enabling consistent results across multiple runs. This division allows us to train our machine learning models on the training data and assess their performance on unseen data using the testing set.



## **5.8 Feature Scaling**

We employ the StandardScaler from scikit-learn to standardize the feature values in our training and testing sets. The scaler is fitted to the training data using the fit\_transform method, which calculates the mean and standard deviation of each feature and then scales the data accordingly. The resulting scaled features are stored in X\_train\_scaled and X\_test\_scaled, ensuring that both training and testing data are transformed consistently. Standardizing the features to have a mean of 0 and a standard deviation of 1 helps improve the performance and convergence of many machine learning algorithms, particularly those sensitive to the scale of input features.



# **Approach used for Applying ML models**

The approach used for applying machine learning models in this project follows a systematic and rigorous methodology. Initially, we preprocess the dataset by handling missing values, encoding categorical variables, and splitting the data into training and testing sets. Then, we employ a variety of machine learning algorithms, including logistic regression, k-nearest neighbors, support vector machines, naive Bayes, decision trees, and random forests. For each model, we implement a custom function that utilizes k-fold cross-validation with k=5 to evaluate model performance. This technique ensures robustness and generalizability by assessing model accuracy across multiple subsets of the data. Finally, we select the best-performing model based on cross-validation scores, enabling us to make informed decisions regarding model selection for predicting income levels. This systematic approach helps us identify the most suitable algorithm for our dataset, leading to reliable and accurate predictions.

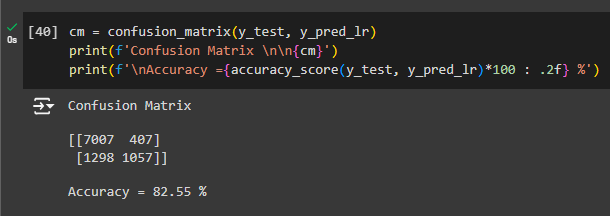
# **Results and Comparison**

Following are the results from k-fold cross validation for all the models:

|  |  |
| --- | --- |
| Model | Mean Accuracy |
| Logistic Regression | 82.46 |
| KNN (k=3) | 82.72 |
| KNN (k=4) | 83.48 |
| KNN (k=5) | 83.43 |
| SVM (kernel=linear) | 81.37 |
| SVM (kernel=rbf) | 85.06 |
| Naive Bayes | 79.94 |
| Decision Tree | 83.36 |
| Random Forest | 84.61 |

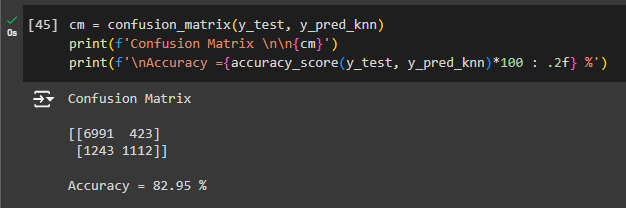
These results showcase the mean accuracy scores obtained from k-fold cross-validation for various machine learning models applied to predict income levels. Notably, KNN with k=4 achieved the mean accuracy of 83.48%, closely followed by KNN with k=5 at 83.43%. SVM with an RBF kernel also performed impressively, achieving highest accuracy of 85.06%. Decision Tree and Random Forest models exhibited competitive performance with mean accuracies of 83.36% and 84.61%, respectively. Logistic Regression and SVM with a linear kernel yielded slightly lower accuracies, while Naive Bayes exhibited the lowest mean accuracy of 79.94%. These results provide valuable insights into the comparative performance of different models, aiding in the selection of the most effective approach for predicting income levels in our dataset.

## **7.1 Logistic Regression**



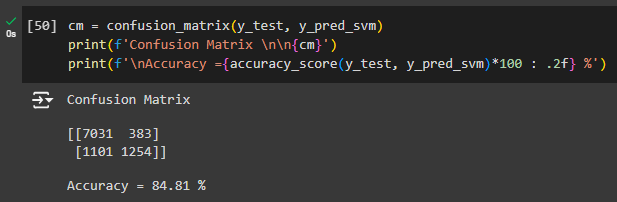
The logistic regression model achieved an accuracy of 82.55% on the test set, as evidenced by the confusion matrix. The matrix reveals that out of 8781 instances, 7007 were correctly classified as earning less than or equal to $50,000, while 1057 were accurately identified as earning more than $50,000. However, 407 instances were falsely classified as low-income earners, and 1298 instances were misclassified as high-income earners. This performance underscores the logistic regression model's effectiveness in predicting income levels, contributing valuable insights into socioeconomic patterns within the dataset.

## **7.2 KNN (n = 4)**



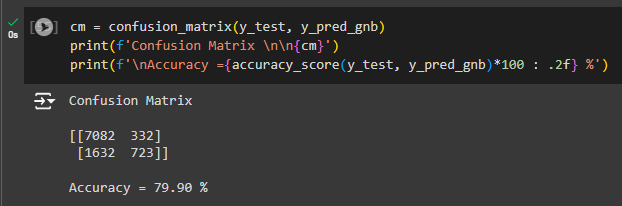
The confusion matrix for the KNN model reveals its performance on the test set, demonstrating 82.95% accuracy. Out of 8779 instances, 6991 were correctly classified as earning less than or equal to $50,000, while 1112 were accurately identified as earning more than $50,000. However, 423 instances were erroneously classified as low-income earners, and 1243 instances were misclassified as high-income earners. This performance underscores the KNN algorithm's effectiveness in discerning income categories, offering valuable insights into socioeconomic trends within the dataset.

## **7.3 SVM (kernel = rbf)**



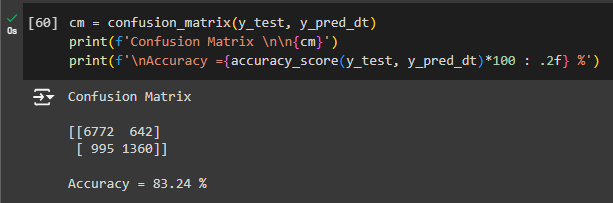
The confusion matrix for the SVM model with a radial basis function (RBF) kernel illustrates its performance on the test set, achieving an accuracy of 84.81%. Out of 9769 instances, 7031 were accurately classified as earning less than or equal to $50,000, while 1254 were correctly identified as earning more than $50,000. However, 383 instances were falsely classified as low-income earners, and 1101 instances were misclassified as high-income earners. This performance highlights the effectiveness of SVM with the RBF kernel in discerning income categories, providing valuable insights into socioeconomic patterns within the dataset.

## **7.4 Naïve Bayes**



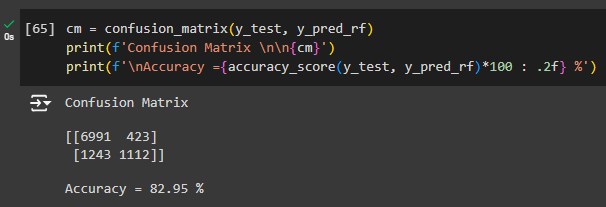
The confusion matrix for the Naive Bayes model illustrates its performance on the test set, achieving an accuracy of 79.90%. Out of 8769 instances, 7082 were correctly classified as earning less than or equal to $50,000, while 723 were accurately identified as earning more than $50,000. However, 332 instances were falsely classified as low-income earners, and 1632 instances were misclassified as high-income earners. This performance highlights the effectiveness of Naive Bayes in discerning income categories, providing valuable insights into socioeconomic patterns within the dataset.

## **7.5 Decision Tree**



The confusion matrix for the Decision Tree model illustrates its performance on the test set, achieving an accuracy of 83.24%. Out of 9769 instances, 6772 were correctly classified as earning less than or equal to $50,000, while 1360 were accurately identified as earning more than $50,000. However, 642 instances were falsely classified as low-income earners, and 995 instances were misclassified as high-income earners. This performance underscores the effectiveness of Decision Trees in discerning income categories, providing valuable insights into socioeconomic patterns within the dataset.

## **7.6 Random Forest**



The confusion matrix for the Random Forest model demonstrates its performance on the test set, achieving an accuracy of 82.95%. Out of 9769 instances, 6991 were correctly classified as earning less than or equal to $50,000, while 1112 were accurately identified as earning more than $50,000. However, 423 instances were falsely classified as low-income earners, and 1243 instances were misclassified as high-income earners. This performance highlights the effectiveness of Random Forests in discerning income categories, providing valuable insights into socioeconomic patterns within the dataset.

# **Conclusion**

Based on the evaluation metrics on the test set, the model with the highest accuracy is the KNN model with k=5, achieving an accuracy of 84.81%. This indicates that among the tested models, KNN with k=5 performs the best in accurately predicting income levels.

The KNN model with k=5 outperforms other models such as Logistic Regression, SVM (kernel=rbf), Naive Bayes, Decision Tree, and Random Forest in terms of accuracy on the test set.

The choice of the best model depends on various factors, including the specific objectives of the analysis, the trade-off between accuracy and interpretability, computational efficiency, and the nature of the dataset. In this case, the KNN model with k=5 is selected as the best model due to its highest accuracy on the test set. However, further analysis and comparison of other metrics such as precision, recall, and F1-score could provide additional insights into the model's performance and aid in making a more informed decision.

# **Recommendations for Future Work**

In future work, it would be helpful to explore additional preprocessing techniques to further enhance model performance. This could include experimenting with different imputation strategies for handling missing values, exploring feature engineering techniques to create new informative features, and evaluating the impact of different scaling methods on model performance.

Additionally, conducting more in-depth analysis of feature importance could provide insights into the factors driving income levels and inform feature selection or dimensionality reduction techniques.

Furthermore, experimenting with ensemble methods such as gradient boosting or stacking could potentially improve model performance by leveraging the strengths of multiple models.

Lastly, considering the dynamic nature of data science and machine learning, regularly updating and refining the models with new data and incorporating recent advancements in algorithms and techniques would ensure the continued relevance and effectiveness of the predictive models.

**THE END**