Adult Income Classification Project Documentation

Goal: Predict whether a person earns more than $50K a year based on their personal, educational, and work-related information using a machine learning model.

Overview:

In this project, we used the Adult Income dataset (also known as the Census Income dataset) to build a classification model that predicts whether an individual's income exceeds $50,000 per year.

The dataset includes attributes such as age, education, job type, marital status, hours worked per week, and more. We processed and engineered features from this data and trained a Random Forest Classifier, a powerful and ensemble-based machine learning algorithm that works well with both numerical and categorical data.

The model is capable of making accurate predictions and can be integrated into a web application for real-time use.

Step 1: Importing the Libraries

In this step, I imported all the required Python libraries like `pandas`, `numpy`, `matplotlib`, `seaborn`, and machine learning libraries from `sklearn`. These are used for handling the dataset, visualizing it, and building the model.

Step 2: Loading the Dataset

I loaded the **Adult Income dataset**, which contains personal and work-related information of individuals. This data helps in predicting whether a person earns more than $50K or not.

Step 3: Checking the Dataset

After loading, I viewed the top rows of the data using `head()` and checked the shape of the dataset to know how many rows and columns are present. This gives a quick idea of the structure and size of the data.

Step 4: Adding Column Names

The dataset didn't come with column headers, so I manually added appropriate column names like `age`, `job_type`, `education_level`, `marital_status`, `income`, etc., to make the dataset readable and easy to use in later steps.

Step 5: Splitting Target and Features

I separated the **target variable** (`income`) from the **features** (all other columns). The target is what we want to predict — whether income is more than $50K or not.

Step 6: Handling Null Values

I checked for missing values in the dataset. Wherever missing or unknown values (like ' ?') were found, I handled them either by removing the rows or by cleaning the data to make it usable for model training.

Step 7: Adding New Features

I created **new columns** to help the model learn better, such as:

- `is_senior` → based on age
- `net_capital` → capital gain - capital loss
- `is_married`, `is_native_us`, `experience_estimate`, and more
  These help capture useful insights from existing data.

Step 8: Rechecking for Missing Values

After adding new features, I again checked for any missing data that may have been introduced and handled it properly to maintain data quality.

Step 9: Data Visualization

I visualized the data using `matplotlib` and `seaborn`. This helped me understand the relationships between variables and see trends — for example, how education or hours worked affects income.

Step 10: Encoding Categorical Features

Since machine learning models don't work with text, I converted all **categorical columns** (like job type, education level, gender) into numerical format using one-hot encoding and label encoding.

Step 11: Data Standardization

I scaled the numerical features using **StandardScaler** so that features like age, capital gain, etc., have similar ranges. This helps the model learn efficiently.

Step 12: Model Training

I trained a **Random Forest Classifier** on the processed dataset. It learned patterns from the data to classify whether a person earns `>50K` or `<=50K`.

Step 13: Model Evaluation

I evaluated the model using accuracy, precision, recall, and confusion matrix. The model gave a good performance on the test set, indicating it learned the data well

Step 14: Model Prediction

I tested the model on a new record to see if it can correctly predict income based on a person's age, job, education, work hours, etc. The prediction was successful.

Step 15: Saving the Model

Finally, I saved the trained model using `pickle`. This allows me to reuse the model without training it again, especially when connecting it with the frontend application.

Step 16: Importing Required Libraries for Deployment

For the deployment part of the project, we imported necessary libraries again — especially for **data preprocessing**, **model loading**, and **handling user input**. This includes libraries like `pickle` (for loading the saved model), and `sklearn` tools (for encoding and scaling).

Step 17: Loading the Trained Model

We used Python's `pickle` library to load the previously trained **Random Forest model** from the `.pkl` file. This model is now ready to take new inputs and make predictions.

Step 18: Loading the Dataset for Taking Input

To ensure consistency, we reused a **copy of the dataset** (structure only, no target column) as a template. This helps us make sure the structure of the user input matches what the model expects.

Step 19: Renaming the Columns

Sometimes, the column names need cleaning or adjusting (like removing spaces or fixing typos). We renamed them properly to match the model's training phase.

Step 20: Taking User Input

Instead of taking manual input from the console or command line every time, we simulated user input by manually feeding the values into the script. Later, this can be connected to a front-end app (like Streamlit or Flask).

Step 21: Triggering Prediction on Button Click

We set a condition — **if the user clicks "Predict"** — to trigger the preprocessing and prediction logic. This part is essential in a deployed app to keep the prediction on-demand.

Step 22: Loading the Scaler

We loaded the **same StandardScaler** object used during training to ensure that the input data is transformed the same way before prediction.

Step 23: Configuring the Input Data with Columns

We converted the user input into a **DataFrame with correct column names**, just like the original dataset. This step helps ensure the model receives data in the correct format.

Step 24: Adding New Features

We added derived features like `is_senior`, `net_capital`, `is_married`, etc., which were also used during training. These help the model make more accurate predictions.

Step 25: Encoding the Categorical Features

Categorical columns like `job_type`, `gender`, and `country_of_origin` were encoded using the **same encoders** used during training. This step converts strings into numerical values the model understands.

Step 26: Scaling the Data

We scaled the final numerical dataset using the previously loaded scaler. This ensures that all features are within the same range, just like during training.

Step 27: Predicting the Income

After preprocessing, we passed the input to the **trained model** to predict whether the income is **more than $50K or not**.

Step 28: Showing the Prediction

Finally, we displayed the predicted class to the user — either **“Income > $50K”** or **“Income ≤ $50K”** — in a user-friendly format.